Eldo® User's Manual

Release AMS 2009.2

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: www.mentor.com/supportnet
Send Feedback on Documentation: www.mentor.com/supportnet/documentation/reply_form.cfm

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction to Eldo

Introduction

The Eldo® analog simulator is the core component of a comprehensive suite of analog and mixed-signal simulation tools. Eldo offers a unique partitioning scheme allowing the use of different algorithms on differing portions of design. It allows the user a flexible control of simulation accuracy using a wide range of device model libraries, and gives a high accuracy yield in combination with high speed and high performance.

Overview

The following is a list of the major product features of Eldo:

- Eldo is the core technology allowing to address RF simulation (Eldo RF) and mixed-signal (Questa ADMS, ADMS-ADiT)
- Simulation of very large circuits (up to around 300,000 transistors) in time and frequency domains
- 3× to 10× gain in simulation speed over other commercial SPICE simulators, while maintaining same accuracy
- Three complementary transient simulation algorithms (OSR, Newton, IEM)
- Flexible user control of simulation accuracy
- Unique transient noise algorithm
- Advanced analysis options such as pole-zero, enhanced Monte Carlo, DC mismatch
- Circuit optimization and statistical analysis (Design of Experiments)
- S and Z-domain generalized transfer functions
- Reliability simulation (Eldo UDRM)
- Flexible DSPF netlist support
- Behavioral modeling with Verilog-A
- Extensive device model libraries including leading MOS, bipolar and MESFET transistor models such as the BSIM3v3.x, BSIM 4, MM11, Mextram, HICUM, and PSP
- TSMC Model Interface (TMI) support

- IBIS (I/O Buffer Information Specification) model support
- Integration into Mentor Graphics IC flow, consisting of Design Architect IC for schematic capture, IC station for the layout side, and Calibre/Calibre xRC for DRC/LVS and extraction. This flow provides a complete, front-to-back design and verification environment for analog, mixed-signal and RF.
- Integration into Cadence's Analog Artist environment (Artist Link)

Eldo Input and Output Files

The following flowchart shows the input files that must be provided for an Eldo simulation run and the output files that Eldo produces:

Figure 1-1. Eldo Input & Output Files

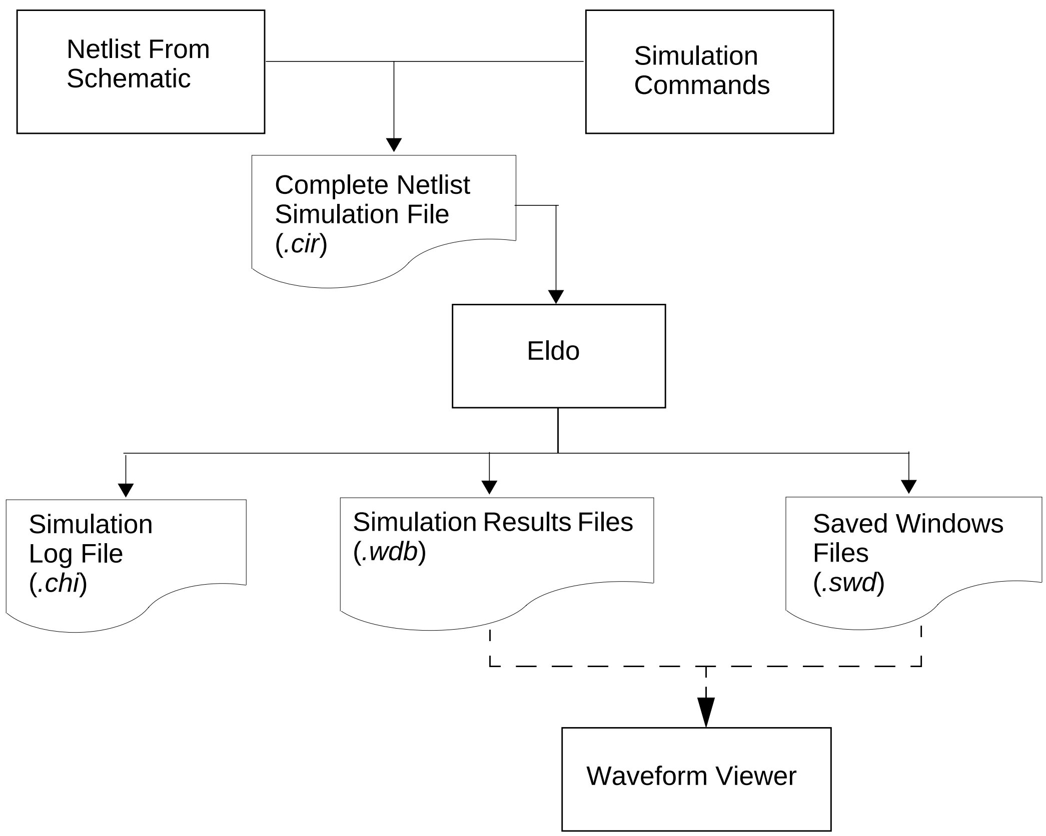


Figure 1-1 shows the main files used by Eldo. A brief description of each is given below:

<file>.cir — The main Eldo control file, containing circuit netlist, stimulus and simulation control commands. This file is SPICE compatible, the Eldo control language being a superset of the Berkeley SPICE syntax.

<file>.chi SPICE compatible output log file containing ASCII data, including results and error messages.

<file>.wdb A binary output file for mixed-signal JWDB format files. This is always generated by default. Viewed with the EZwave waveform viewer. By default, using the *.wdb* format, the **.EXTRACT** and **.MEAS** waveforms are also saved inside the *EXT* folder in the main *.wdb* file. Waveforms defined by **.DEFWAVE** commands combined with **.PLOT** are saved inside the appropriate analysis folder (for example TRAN) in the main *.wdb* file.

<file>.swd A saved windows file used by the EZwave waveform viewer. This file contains information on waveforms and their display and cursor settings, window format settings and complex waveform transition settings.

Other files not generated by default and not shown in the figure are:

<file>.cou A binary output file containing Eldo analog simulation results data. A special interface is provided to access this data from your own post-processor software if required. This is a legacy output format. Please refer to the *Eldo cou Library User's Manual* for more details.

<file>.ext A file containing extraction or waveform information, created when using a **.EXTRACT** command in the netlist and the *.cou* format output is specified. This file will not always be output, it depends on the type of simulation and the specification of the **.EXTRACT** command.

<file>.ext.wdb
A file containing extraction or waveform information, created when using a **.EXTRACT** command in the netlist with option **EXTFILE** specified (using the default *.wdb* format). This file will not always be output, it depends on the type of simulation and the specification of the **.EXTRACT** command.

<file>.meas A file containing extraction or waveform information when the commands **.EXTRACT**, **.MEAS**, or **.PLOT W**(XX) are present in an input netlist and the *.cou* format output is specified. This file will not always be output, it depends on the type of simulation and the specification of the **.EXTRACT**, **.MEAS**, or **.PLOT W**(XX) command.

<file>.meas.wdb
A file containing extraction or waveform information when the commands **.EXTRACT**, **.MEAS**, or **.PLOT W**(XX) are present in an input netlist with option **MEASFILE** specified (using the default *.wdb* format). This file will not always be output, it depends on the type of simulation and the specification of the **.EXTRACT**, **.MEAS**, or **.PLOT W**(XX) command.

Note

The above four types of file (*.ext*, *.ext.wdb*, *.meas*, *.meas.wdb*) are also generated when functions are used in **.DEFWAVE** commands. In most cases, the result of such functions are known only at the end of the simulation, so the waves issued from a **.DEFWAVE** can not be plotted in the *.cou* output file, but only in a specific file generated at the end of the simulation. These types of file are not generated if you don't use **.PLOT** or **.PRINT** commands in the netlist or options **EXTFILE** or **MEASFILE** specified.

<file>.aex A file containing extraction information, created when **.OPTION AEX** is used in conjunction with **.EXTRACT** or **.MEAS** commands.

<file>.pz The output file used by the Pole Zero post-processor.

<file>.spi3 SPICE3 compatible output file.

<file>.wsf Cadence compatible output file.

<file>.fsdb nWave viewer compatible output file.

The *.cou*, *.ext*, *.meas*, *.aex*, *.pz*, *.spi3*, *.wsf*, and *.fsdb* files are only produced by Eldo when the user has set appropriate options in the input file or command line flags.

For more details, refer to Simulator Commands.

Getting Started

How to Run Eldo

To run an Eldo simulation, a *.cir* control file must be supplied to the simulator. As can be seen in the previous example, this file must include the following:

- Circuit connectivity, i.e. a netlist.
- Model parameter values defining the specific device models to be used.
- Electrical stimuli (sources).
- Simulation options and commands.

Input and output formats are compatible with Berkeley SPICE 2G6, however Eldo provides additional features not implemented in SPICE.

Schematic Example

This example consists of a simple cascade of three inverters. The figures below show the circuit diagram for the cascade together with the inverter subcircuit. In order to create the Eldo netlist, node names must be assigned to the circuit. The complete netlist is shown on the following page.

Figure 1-2. Cascaded Inverter Circuit

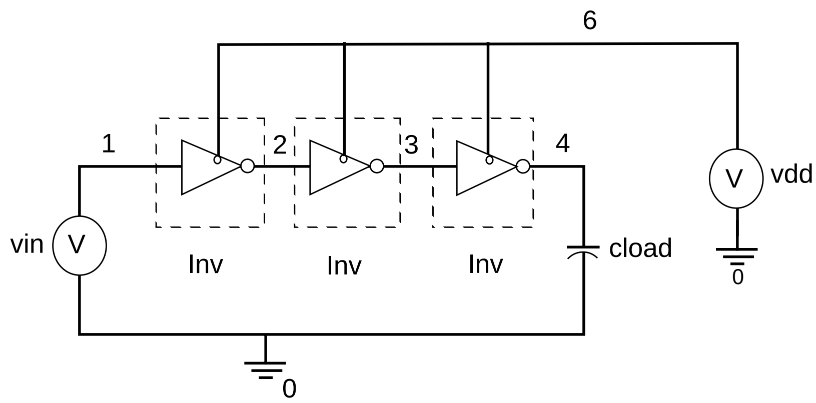
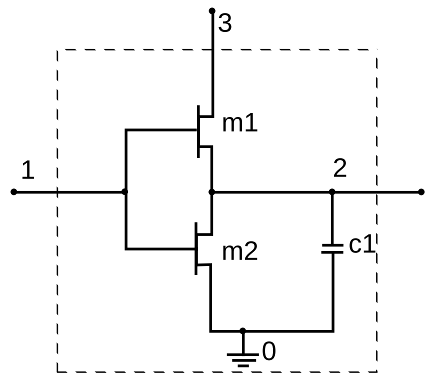


Figure 1-3. Inverter Subcircuit



Associated Netlist

The *.cir* control file for Eldo can be generated using a basic text editor, or alternatively a schematic editor that is capable of generating SPICE-like format.

For further information, please refer to Eldo Control Language.

Sample circuit .cir control file

```
* MOS model definitions
.model m1 nmos level=3 vto=1v uo=550 vmax=2.0e5
+ cgdo=0.4p cgbo=2.0e-10 cgso=4.0e-11 cjsw=10.e-9
+ mjsw=0.3 tox=1.0e-7 nsub=1.0e16 nfs=1.5e10
+ xj=0.5u ld=0.5u pb=0.75 delta=0.9 eta=0.95
+ kappa=0.45 gamma=0.37
.model p1 pmos level=3 vto=-1v uo=230 vmax=1.9e5
+ cgdo=0.4p cgbo=2.0e-10 cgso=4.0e-11 cjsw=10.e-9
+ mjsw=0.3 tox=1.0e-7 nsub=1.0e16 nfs=1.5e10
+ xj=0.5u ld=0.5u pb=0.75 delta=0.9 eta=0.95
+ kappa=0.45 gamma=0.37
* Subcircuit definition
.subckt inv 1 2 3
m2 2 1 0 0 m1 w=10u l=4u ad=100p pd=40u as=100p
m1 2 1 3 3 p1 w=15u l=4u ad=100p pd=40u as=100p
c1 2 0 0.5p
.ends inv
* Subcircuit calls
x1 1 2 6 inv
x2 2 3 6 inv
x3 3 4 6 inv
cload 4 0 1p
* Electrical source definitions
vdd 6 0 5v
vin 1 0 pulse(0 5 10e-9 5e-9 5e-9 30e-9 50e-9)
* Simulation options & commands
.tran 0.5n 100n uic
.ic v(1)=0
.plot tran v(1) v(2) v(3) v(4)
.print tran v(1) v(2) v(3) v(4)
.option eps=0.5e-3 tnom=50 list node
.end
```

Please see “Documentation Conventions” on page 38 for a detailed description on the meanings of the different fonts, brackets, and so on, used throughout this manual.

Running a Simulation

To run a simulation from the command line (see “Running Eldo” on page 41 for a full list of options) use the following command:

```
eldo cir_file_name.cir
```

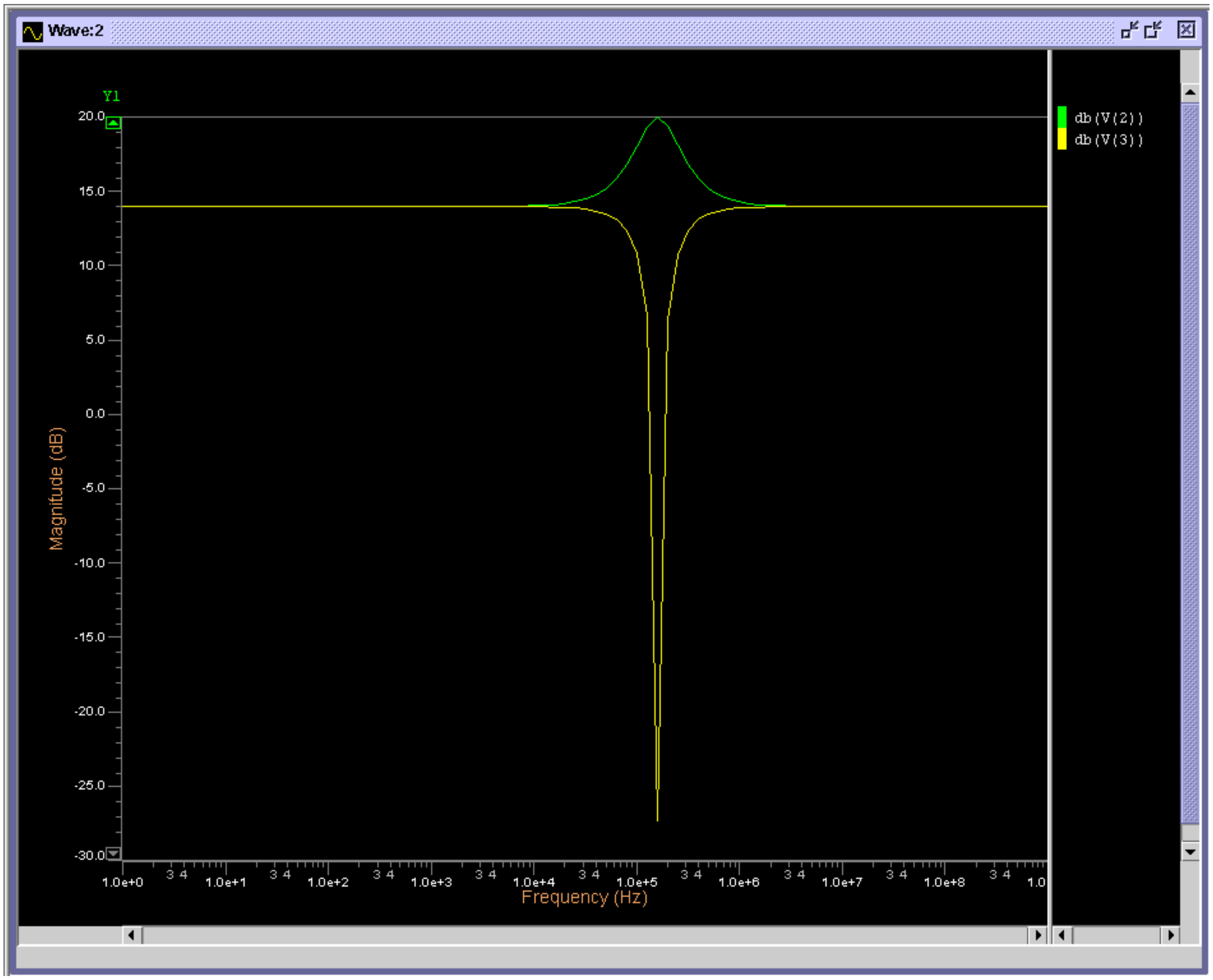
After the simulation has been completed, Eldo writes simulator information to the *.chi* file. This file will contain details of the simulation including any warning or error messages, which may have been encountered during simulation. A binary *.wdb* file is also generated by default as an output of simulation. The user can view the results written to the *.wdb* file, with the EZwave viewer. The *.wdb* file can be opened in the EZwave viewer the using the following command:

```
ezwave cir_file_name.wdb
```

For further information regarding the use of EZwave, please refer to the *EZwave User's Manual*.

See Figure 1-4 for an example binary output file (*.wdb*) viewed with EZwave.

Figure 1-4. EZwave output (.wdb)



Related Documentation

Other documents and manuals that are referenced in this manual, and that you may need to refer to are:

- *Eldo Device Equations Manual* for a complete set of Eldo model equations
- *Eldo RF User's Manual*
- *Eldo UDM User's Manual*
- *Eldo UDRM User's Manual*
- *Eldo Verilog-A User's Manual*
- *EZwave User's and Reference Manual*
- *Questa ADMS User's Manual*
- *ADiT User's Manual*
- *CFAS User's Manual*
- *Eldo cou Library User's Manual*

Documentation Conventions

The following is a list of documentation conventions used throughout the *Eldo User's Manual* both in the syntax and in the syntax descriptions:

Table 1-1. Documentation Conventions

`USER INPUT`	Eldo netlist content is always shown in `courier non-bold` typeface. Eldo is case insensitive.
`KEYWORDS`	Eldo keywords are shown in **`courier bold`** typeface. Eldo is case insensitive. Example: **`NONOISE`** is an Eldo keyword.
`[ ]`	Square brackets indicate that a parameter is optional. Optional parameter declarations may also be nested. Example: `[[`**`DC`**`] DCVAL]` shows that the value parameter `DCVAL` is optional with the possibility of specifying another optional keyword parameter **`DC`**.
`{ }`	Braces (curly brackets) indicate multiple occurrences of a parameter. Example: `NN {NN}` shows that the parameter `NN` may be specified more than once. The parameter `NN` could be a node name for example.

Table 1-1. Documentation Conventions

<table>
<tr><td>{ }</td><td>Certain syntax, for example expressions, have to be enclosed in braces (curly brackets). In these cases, the braces are shown in bold in order to distinguish them from the braces used above to indicate multiple occurrences. They are part of the syntax.
Example: VALUE={2U*V(3, 4)*I(V5)} shows that VALUE is calculated by the expression in braces.</td></tr>
<tr><td>( )</td><td>Certain syntax, for example pairs of values or functions, must be specified inside brackets, often with a comma to separate values from one another. They are shown in BOLD to avoid confusion with other parentheses. They are part of the syntax.
Example: PULSE (V0 V1 [TD [TR [TF [PW [PER]]]]]) shows the parameter specifications for a pulse function with BOLD brackets ( ) to indicate that all the parameters must be enclosed in brackets.</td></tr>
<tr><td>,</td><td>A comma is used to separate pairs of values. It is shown in bold in order to indicate that it is actually part of the syntax which must be specified.
Example: I(Vxx[, Vyy]) shows the syntax for obtaining the current difference between the voltage sources Vxx and Vyy within the .PRINT command.</td></tr>
<tr><td>.</td><td>Used to indicate the start of an Eldo command. All Eldo commands must begin with a dot since it is part of the command syntax. The dot is shown in bold as well as the command since it can be considered to be part of the command keyword.
Example: .MODEL MNAME TYPE [PAR=VAL] shows the syntax for a model declaration.</td></tr>
<tr><td>.</td><td>An unbold dot is used to indicate nodes or subcircuits within other subcircuits.
Example: .plot v(x2.x1.1) shows the plotting of node 1 which is inside subcircuit x1 which is inside subcircuit x2.</td></tr>
<tr><td>< ></td><td>Angle brackets are sometimes used to indicate items which are not part of the syntax, but which are descriptions, and so on.
Example:
...
<CIRCUIT_COMPONENTS>
...
The angle brackets are used to prevent the name CIRCUIT_COMPONENTS from being confused with the Eldo syntax. <CIRCUIT_COMPONENTS> above simply means that between the dots, ..., may be a number of different circuit components.</td></tr>
<tr><td>|</td><td>The logical OR symbol is used to indicate that one of the parameters must be chosen.
Example: R|C|I|V shows that either R, C, I or V must be chosen.</td></tr>
</table>

The conventions described above are used in combination with each other to describe the way in which the syntax has to be specified.

Example

```
.SUBCKT NAME NN {NN} [(ANALOG)] [PARAM: PAR=VAL {PAR=VAL}]
...
<CIRCUIT_COMPONENTS>
...
   .ENDS [NAME]
   .SUBCKT LIB FNAME SNAME [LIBTYPE]
```

Shows the syntax for a subcircuit definition.

Chapter 2
Running Eldo

This chapter is divided into the following sections:

Running Eldo from the Command Line

When invoking Eldo at the command line, the following command line flags can be used:

```
eldo  cir_filename
[-32b] [-64b]
[-adit] [-aditbb]
[-ai veriloga_dir_name]
[-alter alter_name | alter_index]
[-ascii]
[-b]
[-checksyntax]
[-clean]
[-cntthread]
[-compat] [-compmod] [-compnet]
[-convert_iic src_file dest_file]
[-cou]
[-cou47]
[-couext]
[-createoutpath output_dir_name]
[-dbledefs]
[-dbp]
[-d | -define macro]
[-E] [-EE]
[-eil file]
[-extract filename]
[-ezwave]
[-float]
[-gwl gwl_lib_name]
[-h hostname]
[-hdlpath veriloga_path]
[-help [commands|devices|sources|manual]]
[-i cir_filename]
```

```
[-inter]
[-isaving [val]]
[-jwdb_config swd_filename]
[-jwdb_extensions]
[-jwdb_nocomplex]
[-jwdb_norffolder]
[-jwdb_servermode]
[-jwdb_threshold [val]]
[-l log_filename]
[-lib object_library_dir_name]
[-libinc]
[-m subckt_name]
[-m53]
[-mgls_async]
[-mthread]
[-noascii]
[-nocatmx]
[-nochi]
[-noconf]
[-nocouext]
[-noerrmlog]
[-nogwl]
[-noinit]
[-nojwdb]
[-no_sst_mthread]
[-nostver]
[-o output_filename]
[-opseldo_jwdb_run]
[-out out_filename]
[-outname out_filename]
[-outpath output_dir_name]
[-ovstatus]
[-parse_only]
[-plspath dir_name]
[-probeop2]
[-queue]
[-restart ["[save_file] [file=wfile]"]]
[-ri]
[-savetime time]
[-searchpath path_list]
[-silent]
[-sp spectre_file [-i spice_command_file] [-s2emode 2|1] [-clean_sp]
   [-spectre_out pathname] [-sp_plot 2|1|0]]
[-spiout]
[-spmodel <subckt_name1> -spmodel <subckt_name2> ...]
[-ssh hostname]
[-ssim]
[-stat [-statfile filename]]
[-stver]
[-tuning [FAST|STANDARD|ACCURATE|VHIGH] -tuning [BACKANNOTATE]]
[-use_proc n | HALF | MAX k | ALL | HYPER]
[-useht]
[-usethread val]
[-v]
[-vamodel <mod_name1> -vamodel <mod_name2> ...]
[-verbose]
[-vlac]
[-wB cou_filename]
```

[-wdl_timeout time]

It is also possible to create a file named *eldo.ini* which will be interpreted and loaded at the very beginning of each simulation. "`Loading eldo.ini`" is displayed whenever a valid *eldo.ini* file is found. See "Eldo Initialization File" on page 60 for further details.

- `cir_filename`

  Name of the *.cir* control file to be simulated. Default extension is *.cir*. By default, the first parameter in the command line is taken to be the *cir_filename* by default.

Caution

The *cir_filename* is mandatory. All other flags are optional.

The function of the command line flags is as follows:

- `-32b`

  Runs the simulation in 32-bit mode on a 64-bit machine. This could speed-up simulations that require small amounts of memory.

- `-64b`

  Runs the simulation in 64-bit mode (available for Solaris and Linux platforms that have had 64-bit OS installed). This enables simulation of circuits which would require more than 2GB of memory, and which would therefore not work on 32-bit machines.

  To run Eldo with the -64b flag on a Solaris 64-bit machine you must first set the environment variable *AMS_VCO_MODE* to 64, for example:

  ```
  setenv AMS_VCO_MODE 64
  ```

Note

Only a subset of Eldo/Eldo RF is ported. The following features are not supported: SimPilot interface, HDL-A, and output file formats other than *.cou*/*.wdb* file.

- `-adit`

  Sends blocks to ADiT instead of Eldo. Can be overridden by the netlist partitioning commands. Once Eldo detects the -adit flag, it executes the partitioning directives and sends any ADiT designated blocks to ADiT. What is left will be simulated in Eldo.

- `-aditbb`

  As above, but sends blocks to ADiT in black-box mode without first parsing them.

- `-ai verilogа_dir_name`

  Defines a repository directory for compiled Verilog-A models (*.ai*) and compilation files (*.log* and *.info*). If the specified directory does not exist, then it is created. By default, without this flag, Eldo calls the Verilog-A compiler without any argument, thus *.ai* files are created in the working directory. This argument has precedence over netlist instructions. This argument applies to both **`.verilog`** and **`.use_verilog`** commands and is only taken into account when **`-vlac`** is specified.

The name of the output files will be extracted from the **`.VERILOG`** command. For example:

```
.VERILOG -o dummy/module.ai resistor.va
eldo -vlac -ai test ...
```

Eldo will create output file *test/module.ai*.

- `-alter alter_name | alter_index`

  Allows running one specific **`.ALTER`** section by specifying it's name or index, without first running the main netlist. The *.chi* file and std output indicate that Eldo is only running a specific **`.ALTER`** due to this command line flag. Specifying an alter_index of 0 means the normal run without any alters; everything in the netlist is simulated except the modified re-run alter statements.

- `-ascii`

  Forces Eldo to store the results including the printing of the output curves in the ASCII output *.chi* file, see also "ASCII" on page 998. The size of this file can increase significantly for large simulations when these results are printed inside it.

- `-b`

  Runs the simulation in the background, batch mode. Eldo will return the command prompt during simulation. The simulation cannot be terminated using Ctrl+C. A log of the simulation will be written to a *.log* file which can be specified with the `-l log_filename` flag. You can set the *AMS_USER* environment variable to an email address to receive an email notification once the simulation has finished.

- `-checksyntax`

  Forces Eldo to only parse the netlist and elaborate the design without performing simulation. Eldo will perform a syntax check of the netlist and generate warnings and errors as necessary.

- `-clean`

  Remove all old intermediate files for the simulation *filename* specified. Useful for cleaning up old intermediate simulation files left around after a crashed simulation.

- `-cntthread`

  Checks how many CPUs Eldo can access on a multi-processor machine when using `-mthread`. Returns details about the computer architecture. Eldo will share computer resources for multi-threading a single DC or TRAN simulation.

  See "Multi-Threading Eldo Simulations" on page 58 for a full description of multi-threading in Eldo.

- `-compat`

  For simulator compatibility the `-compat` runtime flag can be specified, or alternatively the option **`COMPAT`** command. Provides HSPICE compatibility. When Eldo is invoked with this argument, the main effect is that it accepts some HSPICE syntax. Full details are provided where applicable throughout this manual. For more information and a full list of the effects this flag/command option has, see "HSPICE Compatibility" on page 1373.

  Flag `-compat` is equivalent to setting both `-compmod` and `-compnet` flags shown below:

- `-compmod`

  Triggers only the automatic conversion of models (can alternatively be set with option **`COMPMOD`**). Provides HSPICE models compatibility.

- `-compnet`

  Causes the netlist to be interpreted as compatible format, but the models themselves are treated as Eldo SPICE models (can alternatively be set with option **`COMPNET`**). Provides HSPICE netlist compatibility.

For more information and a full list of the effects the `COMPxxx` flags/command options have, see "HSPICE Compatibility" on page 1373.

- `-convert_iic src_file dest_file`

  Converts a compressed file into a full ASCII file (names expanded and double values stored in ASCII). *cir_filename* should not be specified with this argument. *src_file* is the name of the file with information saved using either the **`.SAVE`** or **`.TSAVE`** command, a *.iic* file is the default extension when saving a simulation run. *dest_file* is the destination ASCII filename. If *dest_file* already exists, the user is warned and prompted whether or not to overwrite it.

- `-cou`

  Generate output in binary Cou format file, see also "`COU`" on page 1011. This can also be specified using the invoke command `-gwl cou`. When JWDB output is not disabled, the *.cou* database will only contain real and imaginary parts of complex waveforms. To avoid this limitation, use `-jwdb_nocomplex` or `-nojwdb`.

- `-cou47`

  Forces Eldo to create a different *.cou* file format (4.7). This format allows faster loading of a large database with many curves and/or several simulations from which only a few number of curves should be loaded. Only for use when cou output format is specified.

  Refer to the *Eldo cou Library User's Manual* for further information.

- `-couext`

  Merge extractions. Forces Eldo to dump **`.EXTRACT`**/**`.MEAS`** information into the *.wdb* or *.cou* file rather than in the *.ext*/*.meas* file. See also `-nocouext`.

- `-createoutpath output_dir_name`

  Directory in which all output files are created. If the output directory specified does not exist, it is created. If not specified, output files are created in the same directory as the *.cir* input file. See also `-outpath`.

- `-dbledefs`

  Allows several definitions of the same **.SUBCKT** or of the same model to be accepted. This works only when Eldo is called using the `-parse_only` flag.

- `-dbp`

  Forces AC output to be (**DB,PHASE**) by default. See also `-ri`.

Note

The command **VDB(**x**)** will always dump **VDB(**x**)**. Flag `-dbp` (and `-ri`) apply to **.PROBE AC V(**X**)** for instance.

- `-dump_file_list file`

  Generates a file containing all the files opened by Eldo (except temporary Eldo files) for a simulation. Useful for packaging an Eldo testcase. If a filename is not specified, then the netlist name is used with the extension *.files_dump*. If a relative path is specified then the file will be generated from the directory specified in `-outpath output_dir_name`.

  The generated file contains all the absolute names of the opened files and is sorted by path and then by name (case ignored). Only one file is generated even if there are **.ALTER** statements or multiple netlists in the source file.

  This can also be specified through an option, see also "DUMP_FILE_LIST" on page 1011.

- `-d | -define macro`

  Define a macro at invoke time. Allows the user to define variables used by the pre-processor. See "Directives Interpreted by the Eldo Parser (Default)" on page 75 and "Directives Interpreted using the C Pre-Processor (-E/-EE Arguments)" on page 77.

- `-E`

  Allows extended use of pre-processor commands to define macros and replace them inside the netlist. Only the main netlist is sent to the C pre-processor. See "Directives Interpreted using the C Pre-Processor (-E/-EE Arguments)" on page 77.

- `-EE`

  Allows extended use of pre-processor commands to define macros and replace them inside the netlist. The main netlist and all include files are sent to the C pre-processor. See "Directives Interpreted using the C Pre-Processor (-E/-EE Arguments)" on page 77.

- `-eil file`

  Used to invoke Eldo in the interactive mode. Commands will be read from the specified file at invocation.

Further information on the interactive mode can be found in "Eldo Interactive Mode" on page 1483.

- `-extract filename`

  Activates extraction mode (perform measurements using waves stored in *.cou* or *.wdb* files). Eldo only performs the **`.EXTRACT`** commands, without performing the simulation(s). Eldo will decorrelate the *simulation* from the *extraction*. `filename` is the name of an EZwave *.wdb* output file created from a previous simulation using the **`.PLOT`**/**`.PROBE`** command. Required. The extractions in the netlist will be solved using the corresponding waves contained in this file. Multiple levels of extraction are not supported with the `-extract` flag. This functionality can also be specified with the netlist command **`.EXTMOD`**, see ".EXTMOD" on page 636.

- `-ezwave`

  Displays Eldo simulation results in the EZwave waveform viewer while the simulation is running (marching waveforms). This option automatically enables option `-gwl jwdb` if required. Eldo will plot the waveforms as defined in the netlist. When the simulation is complete, Eldo exits but EZwave remains displayed.

- `-float`

  Forces Eldo to create *.wdb*/*.cou*/*.fsdb* files containing FLOAT rather than DOUBLE values, which results in saving 50% of the disk space. This is useful when it is known that the output file will be very large. However, for FFT, floating numbers are sometimes inappropriate as they cause a loss in accuracy. Note this flag was previously named `-couf`.

- `-gwl gwl_lib_name`

  Forces Eldo to generate output in the required format (JWDB, and so on).

  `-gwl jwdb`

  Generate JWDB format files (extension *.wdb*). This is always generated by default, see also "JWDB" on page 1012.

  `-gwl cou`

  Generate binary Cou format file, see "COU" on page 1011. When JWDB output is not disabled, the *.cou* database will only contain real and imaginary parts of complex waveforms. To avoid this limitation, use `-jwdb_nocomplex` or `-nojwdb`.

  `-gwl csdf`

  Generate CSDF format file. CSDF is the Common Simulation Data Format, see "CSDF" on page 1011.

  `-gwl fsdb`

  Generate FSDB format for nWave, see "FSDB" on page 1011.

  `-gwl psf`

  Generate binary Cadence format (used with Artist Link), see "PSF" on page 1014.

`-gwl psfascii`

Generate Cadence format in ASCII, see "PSFASCII" on page 1015.

`-gwl psfop`

Generate output for OP data in binary Cadence format (used with Artist Link). Can be used if PSF files are required for back-annotation of the OP results.

`-gwl psfasciiop`

Generate output for OP data in ASCII Cadence format (used with Artist Link). Can be used if PSF files are required for back-annotation of the OP results.

`-gwl wdf`

Generate WDF format for SPICE Explorer viewer, see "FSDB" on page 1011. This format is available only for Linux 32-bits and Sun 32-bits.

- `-h hostname`

  Run the Eldo simulation using rsh (remote shell) on the machine (CPU) specified by `hostname`. Eldo recognizes the environment variable *MGC_DESIGN_KIT* (defined by IC Flow tools) if set, and uses it in remote simulations.

- `-hdlpath veriloga_path`

  Specifies the search path for Verilog-A files. Used in conjunction with the **`.HDL`** and **`.VERILOG`** commands. See ".HDL" on page 678 and ".VERILOG" on page 933 for further information. The directory search order for Verilog-A files is:

  - current directory
  - path defined by the argument of `-hdlpath`

  Example:

  ```
  eldo test.cir -hdlpath ./lib/veriloga_src
  ```

- `-help [commands|devices|sources|manual]`

  Online help. See Command Line Help.

- `-i cir_filename`

  Input *.cir* file name. The *.cir* input file is mandatory and so this flag identifier is optional. By default, the first parameter in the command line is taken to be the *cir_filename*.

- `-inter`

  Used to invoke Eldo in the interactive mode. Commands are sent interactively instead of sending the commands in the netlist.

- `-isaving [val]`

  Specifies the "spill" threshold. Loads Eldo in incremental saving mode. Incremental saving allows the user to save the waveform data to a *.wdb* file when the Joint Wave DataBase (JWDB) reaches the threshold specified (in bytes). The default value is 100 MB. Identical to `-jwdb_threshold` except for the default unit.

- `-jwdb_config swd_filename`

  The *.swd* (saved window database) file specified in this argument may be the result from a previous EZwave session. The page composition of the netlist will be replaced by the one specified in the *.swd* file. Waveforms resulting from expressions or measurements will be “replayed” using new waves.

- `-jwdb_extensions`

  Enables modifications in the way results are stored inside *.wdb* files. The compound waveform is not created when **`.MC`** `irun=X` is used (since this is a single simulation), this can alternatively be set with option **`JWDB_EXTENSIONS`**`=1`, see “`JWDB_EXTENSIONS=0|1`” on page 1001).

- `-jwdb_nocomplex`

  Disable the generation of complex waves (can alternatively be set with option **`NOWAVECOMPLEX`**).

- `-jwdb_norffolder`

  Disable storing RF results in separate folders within the JWDB database. Instead results will be added to the AC and TRAN folders.

- `-jwdb_servermode`

  Specifies the JWDB server launched by Eldo can be re-used by other simulations. Useful data is stored in a file pointed to by the environment variable *AMS_WDBSERVER_INFO*, its default is *$HOME/.ezwave/jwdbserver.info*. In this mode, the JWDB server will exit after the time specified by this environment variable if Eldo is not using it. Default is 60 minutes. See also Eldo Simulation in the *EZwave User’s and Reference Manual*.

- `-jwdb_threshold [val]`

  Loads Eldo in incremental saving mode. Incremental saving allows the user to save the waveform data to a *.wdb* file when the Joint Wave DataBase (JWDB) reaches the threshold specified (in Megabytes). The default value is 100 MB. Identical to `-isaving` except for the default unit.

- `-l log_filename`

  Specifies the log file name for a background, batch mode simulation (`-b`). If not specified, the log file is set to *eldo_<PID>.log*, where *PID* is the process identifier. Most commonly used in batch mode, but can also be specified in normal simulation mode.

- `-lib object_library_dir_name`

  Name of the directory containing Eldo object library files for dynamic linking. This option is typically only used for Eldo-UDM/GUDM/UDRM analog behavioral modeling. Eldo will search for the specified directory and then search for Eldo library files (*libeldo**) inside this directory, if either of these are not found Eldo will issue a warning. If not specified, the directory is set to: *$MGC_AMS_HOME/$AMS_VCO/lib* where *$AMS_VCO* is a runtime environment variable referring to the platform type, for example, *ixl* for Linux 32-bit.

- `-libinc`

  Specifies that the full contents of every library are read by Eldo in one pass upon completion of reading the input file. This was the default mechanism in the v5.8 version of Eldo. All the libraries (**`.LIB`**) are included without filtering the objects (model, card, or subcircuit) that are not used in the specific netlist. See "LIBINC" on page 966.

- `-m subckt_name`

  Macro simulation. Specifies that all the devices at the top of the circuit (except voltage and current sources) are ignored, and the subcircuit specified `subckt_name` becomes the top-level of the circuit. This can be useful for subcircuit testing.

- `-m53`

  Specifies the ruling surrounding the M factor should be as it was for versions of Eldo before and including v5.3, see "M53" on page 979.

- `-mgls_async`

  Allows asynchronous communication between the MGLS license manager and Eldo. Only use this argument if Eldo hangs at the end of a simulation on a multi-processor machine.

Note

For safety, this argument enables the license manager to queue the license requests when needed. Therefore, if Eldo attempts to checkout licenses faster than they are checked in, Eldo will wait until new licenses are available.

- `-mthread`

  Activates multi-threading for a single DC or TRAN simulation. Eldo will share computer resources on a multi-processor machine. Eldo will make use of all the possible CPUs on the machine. See also option "MTHREAD" on page 954.

  See "Multi-Threading Eldo Simulations" on page 58 for a full description of multi-threading in Eldo.

- `-no_nominal_alter`

  Bypass the "nominal" run in a netlist containing **`.ALTER`** commands, can also be specified with .option NO_NOMINAL_ALTER.

- `-no_sst_mthread`

  Deactivates multi-threading for a Steady-State simulation. See also Multi-Threaded Simulation Options (**`SST_MTHREAD`**) of the *Eldo RF User's Manual*.

- `-noascii`

  Prevents plot/print output waveform information from being produced in the ASCII output (*.chi* file), see also "NOASCII" on page 1004. The file is still created, but the size can be significantly reduced for large simulations when these results are not printed inside it. Eldo default behavior is with option **`NOASCII`** set so this flag should not be required.

- `-nocatmx`

  Disables merging devices in parallel. See also option “NOCATMX” on page 989. For further details see “Merging Devices in Parallel” on page 120.

- `-nochi`

  Prevents ASCII output from being produced (no *.chi* file is produced).

- `-noconf`

  No confirmation. When stopping a simulation run with Ctrl-C, then Eldo exits with no confirmation. Also, it will not ask whether or not to re-enter FAS debugger, if necessary. This option is used in the Artist Link integration. Eldo automatically creates a save file when interrupted whenever a signal interrupt is received by Eldo, even if Eldo does not run in interactive mode (such as when running with LSF or Sun Grid).

- `-nocouext`

  Forces Eldo to dump **.EXTRACT**/**.MEAS** information into separate files *.wdb* (or *.cou*) and *.ext*/*.meas* files rather than merged. See also `-couext`.

- `-noerrmlog`

  Disables the creation of the error message manager log file (named *<cirfile>.errm.log*). This file contains the warning/error messages which are used by the error message manager.

- `-nogwl`

  All output files are generated except *waveform* files (*.cou*, *.wdb*, *.tr0*...)

- `-noinit`

  Disable loading the *eldo.ini* file.

- `-nojwdb`

  Disable the creation of *.wdb* files (can alternatively be set with option **NOJWDB**).

- `-nostver`

  Disables the STMicroelectronics version of Eldo. Further information can be found in “STMicroelectronics Models” on page 1509.

- `-o output_filename`

  Output *.chi* file name. If not specified, the *.chi* file is set to *cir_filename.chi*. *output_filename* has to be the full pathname of the ASCII output file. If *output_filename* does not contain any ‘/’ character (if *output_filename* is just a filename with no path), then any specified *output_dir_name* string will be used to create *<output_dir_name>/<output_filename>*.

- `-opseldo_jwdb_run`

  Organizes the Waveform List of EZwave in a different way allowing easy access to results for different optimizer runs (can alternatively be set with option **OPSELDO_JWDB_RUN**, see “OPSELDO_JWDB_RUN” on page 1009).

- `-out out_filename`

  Specifies the file name for all simulation output files. If not specified, files will use the default of *cir_filename*. A path can be defined as long as the directory exists. By default, the files are created in the same directory as the *.cir* file.

- `-outname out_filename`

  Specifies the file name for simulation output files (*.swd*, *.wdb*, *.cou*). If not specified, files will use the default of *cir_filename*. A path can be defined as long as the directory exists. By default, the files are created in the same directory as the *.cir* file.

- `-outpath output_dir_name`

  Directory in which all output files are created. If this flag is not specified, output files are created in the same directory as the *.cir* input file. If the output directory specified does not exist, it is not created, use `-createoutpath` instead. The final character of *output_dir_name* must be a forward slash '/'. If omitted, this will be added automatically.

- `-ovstatus`

  Causes Eldo to dump at the end of the standard output (stdout) transcript some lines summarizing the status of the simulation. For example:

  ```
  Simulation finished.
  There are 3 parsing warning(s).
  There are 1 simulation error(s).
  ```

  The first line indicates the simulation has finished and the output file transcript is complete, with or without errors and warnings. The second line indicates the number of parsing errors or warnings. The third line indicates the number of simulation errors or warnings.

  This can be useful if using a browser to analyze the status of a run. If there are non-convergence errors, then in addition to the above lines, an extra line will be displayed:

  ```
  including %ld simulation non-convergence error(s).
  ```

  where %ld is the number of errors.

- `-parse_only`

  Forces Eldo to only parse the netlist, generate warning and errors if any, and exit before the generation step (elaboration of the design).

- `-plspath dir_name`

  Directory in which *.pls* files, generated by S-parameter file conversion, are created. By default, *.pls* files are created in the same directory as the *.s4p* or *.par* files. See "Working with S, Y, Z Parameters" on page 1041.

- `-probeop2`

  Eldo will generate a *circuit.opx* file, where `x` is the index of the run, in which DCOP information will be dumped. This file is an ASCII file which is read by the DA-IC environment. The content of this file is similar to that created when option **`PROBEOP`** is used, but there is some additional information which is dumped: temperature, node information, and now current out of X leaf-instances (leaf-cells are instances which do not call any other X instances).

- `-queue`

  If there is no license available for Eldo, the job will be queued. Used for batch runs.

- `-restart ["[save_file] [file=wfile]"]`

  Restarts a simulation run with information previously saved using the **`.SAVE`** command. The quotes are mandatory if more than one argument is specified. See “.RESTART” on page 854 for further information.

- `-ri`

  Forces AC output to be (**`REAL,IMAG`**) by default. See also `-dbp`.

Note

The command **`VDB(`**`x`**`)`** will always dump **`VDB(`**`x`**`)`**. Flag `-ri` (and `-dbp`) apply to **`.PROBE AC V(`**`X`**`)`** for instance.

- `-savetime time`

  Creates a *.sav* file which saves the context of a simulation every `time` hour of CPU time used in a transient analysis. If the system crashes during a run, the simulation may be restarted from the last saving time. See “.SAVE” on page 857 for further information.

- `-searchpath path_list`

  Libraries and include files are searched inside the specified path when not found. There is no limit on the number of search paths that can be used. This has the same effect as **`SEARCH=`**`path1` option, see “SEARCH=path1[{:path2}]” on page 1022.

- `-silent`

  Disables all displays to the standard output.

- `-sp spectre_file`

  Use a Spectre netlist as input. Eldo will call the *spect2el* script to convert netlists from Spectre format (Spectre language syntax) into Eldo format (Eldo syntax). *spect2el* generates several files that are used to map the Spectre name to a SPICE name. This will also avoid the reconversion of the Spectre files if nothing has changed in them. *spectre_file* must be specified with an extension. No guess is made on the possible extension. If the filename is specified without its extension it will lead to an error.

  See “Spectre Compatibility” on page 1395 for further information.

  `-i spice_command_file`

  Used with the `-sp` argument when a Spectre netlist is the input. *spice_command_file* must be specified with an extension. No guess is made on the possible extension. If the filename is specified without its extension an error will be generated. The *spice_command_file* must not include the *spectre_file*. Eldo will automatically perform the link between these files. Including the Spectre file will not work and the simulation will stop.

`-s2emode 2|1`

Used to switch between the old (pre-2009.1) and new converter. Optional. Used with the `-sp` argument when a Spectre netlist is the input. Set to 1 to run the old converter. Set to 2 (default) to run the newer one, which provides speed and robustness advantages. If errors occur when `-s2emode 2` is specified, a file named *spect2el.error* will be generated clearly describing the errors.

`-clean_sp`

Used with the `-sp` argument when a Spectre netlist is the input. Removes all the files generated by *spect2el*. As a consequence, if you want to relaunch the simulation on that design, the converter will reproduce the work. This will increase the overall simulation time.

`-spectre_out pathname`

Define the path in which the files generated by *spect2el* reside. Optional. Used with the `-sp` argument when a Spectre netlist is the input. Eldo creates *pathname* automatically if it does not exist. If `-clean_sp` is also specified, Eldo clears the files/sub-directories under *pathname*, while *pathname* itself is not removed after simulation. By default, *pathname* is identical to the one specified by `-outpath`.

`-sp_plot 2|1|0`

Controls the conversion of Spectre `save` statements into Eldo **`.PRINT`**, **`.PLOT`** and **`.PROBE`** commands as below:

if `-plot 0`, the converter will convert the `save` statement into **`.PRINT`** (default).
if `-plot 1`, the converter will convert the `save` statement into **`.PLOT`**.
if `-plot 2`, the converter will convert the `save` statement into **`.PROBE`**.

- `-spiout`

  Causes the netlist to be printed without any line numbers in the output file.

- `-spmodel <subckt_name1> -spmodel <subckt_name2> ...`

  Used in conjunction with the **`.HDL`** command. The subcircuit name specified on this flag has higher priority than Verilog-A modules of the same name when both exist. Only one subcircuit name can be specified on each `-spmodel` flag. Multiple names require multiple flags. See also "SPMODEL=subckt_name" on page 1020. See ".HDL" on page 678 for further information. For example:

```
.hdl opamp.va
.subckt myopamp a b c
....
.ends
x1 1 0 2 myopamp

eldo test.cir -spmodel myopamp
```

If the Verilog-A file *opamp.va* contains a module named myopamp Eldo will use the subckt definition instead of the Verilog-A module of the same name. In this configuration the following command is equivalent:

```
eldo test.cir
```

Eldo will first look first for a subckt definition for the X instances. If one is not found or if a vamodel is provided then Eldo will look for Verilog-A modules.

- `-ssh hostname`

  Run the Eldo simulation using ssh (secured shell) on the machine (CPU) specified by `hostname`. Eldo recognizes the environment variable *MGC_DESIGN_KIT* (defined by IC Flow tools) if set, and uses it in remote simulations.

- `-ssim`

  Runs the Motorola (SSIM model) version of Eldo. See also "MOTOROLA" on page 950. This model is only supported on Solaris platforms in Eldo 32-bit mode. Further information can be found in "Motorola SSIM Model (Eldo Level 54 or SSIM)" on page 276.

- `-stat [-statfile filename]`

  Writes design, elaboration, and simulation information to a statistics file. The statistics file is a summary of the simulation activity and is usually quite small. The file should be sent to a support engineer to aid debugging of a design in the event of a problem. The information will not be written to the file if an error occurs during elaboration. For a description of the statistics file, see "Statistics File" on page 63.

- `-stver`

  Runs the STMicroelectronics version of Eldo. Further information can be found in "STMicroelectronics Models" on page 1509.

- `-tuning [FAST|STANDARD|ACCURATE|VHIGH] -tuning [BACKANNOTATE]`

  Selects the default mode of operation of Eldo as regards to precision and speed, see "TUNING=[FAST|STANDARD|ACCURATE|VHIGH] [TUNING=BACKANNOTATE]" on page 972. The `-tuning` flag overrides any **`TUNING`** option settings inside the netlist.

  The `-tuning` flag only accepts one argument at a time, so the option has to be repeated when combining arguments, for example:

  ```
  -tuning ACCURATE -tuning BACKANNOTATE
  ```

- `-use_proc n | HALF | MAX k | ALL | HYPER`

  Activates multi-threading for a single DC or TRAN simulation. Eldo will share computer resources on a multi-processor machine depending on the user-specification below.

  `n`

  use n processors

  `HALF`

  use half the processors

`MAX k`

use a maximum of k processors

`ALL`

use all processors except hyperthreading

`HYPER`

use all processors including hyperthreading processors

See "Multi-Threading Eldo Simulations" on page 58 for a full description of multi-threading in Eldo.

- `-useht`

  Activates hyper-threading to be taken into account when counting the number of processors. By default, Eldo does not take hyper-threading into account, because it is not always efficient to do so. Must be used in conjunction with multi-threading, see `-mthread`.

  See "Multi-Threading Eldo Simulations" on page 58 for a full description of multi-threading in Eldo.

- `-usethread val`

  Activates multi-threading for a single DC or TRAN simulation. Eldo will share computer resources on a multi-processor machine. Eldo will make use of the number of CPUs specified with this flag. This is useful if you want to use a smaller number of threads than the number of available CPUs. For example on a 16 CPU machine, you might only want to use six. The number specified can exceed the number of CPUs available, but this is not recommended. See also "USETHREAD=VAL" on page 962. `-usethread` takes priority over `-mthread` when both flags/options are specified.

  See "Multi-Threading Eldo Simulations" on page 58 for a full description of multi-threading in Eldo.

- `-v`

  Returns the software version number. No simulation is performed.

- `-vamodel <mod_name1> -vamodel <mod_name2> ...`

  Used in conjunction with the **`.HDL`** command. The Verilog-A module name specified on this flag has higher priority than a subcircuit of the same name when both exist. Only one module name can be specified on each `-vamodel` flag. Multiple names require multiple flags. See also "VAMODEL=mod_name" on page 1020. See ".HDL" on page 678 for further information. For example:

```
.hdl opamp.va
.subckt myopamp a b c
....
.ends
x1 1 0 2 myopamp

eldo test.cir -vamodel myopamp
```

If the Verilog-A file *opamp.va* contains a module named myopamp Eldo will use the Verilog-A module myopamp instead of the subckt definition of the same name.

- `-verbose`

  Forces Eldo to display more detailed reporting with some information messages in the standard output terminal. See also "VERBOSE" on page 997. Eldo will print hints about syntax which is valid but ignored if the appropriate analysis is not found in the netlist. For example:

```
Warning 10001: No optimization command has been found in the netlist.

As a consequence:
    1) .option OPSELDO_NETLIST is ignored
    2) .PARAMOPT are interpreted like .PARAM using the initial value,
       or ignored if no initial value has been specified.
    3) GOAL=MINIMIZE is ignored on measurement FOO
    4) GOAL is ignored on measurement FOO2
    5) UBOUND is ignored on measurement FOO3
    6) GOAL=MAXIMIZE is ignored on measurement FOO4

Warning 10002: COMMAND .MC has not been found in the netlist.
  As a consequence:
    1) .option DISPLAY_CARLO is ignored
```

- `-vlac`

  Run the pre-2009.1 Verilog-A compiler. Refer to the *Eldo Verilog-A User's Manual* for further information.

- `-wB cou_filename`

  Output *.cou* file name. *cou_filename* has to be the full pathname of the binary cou output file. If *cou_filename* does not contain any '/' character (if *cou_filename* is just a filename with no path), then any specified *output_dir_name* string will be used to create <*output_dir_name*>/<*cou_filename*>. This flag should not be used with JWDB output.

- `-wdl_timeout time`

  Instructs Eldo to wait the specified `time` seconds for a response from the JWDB server. By default, the timeout is 120 seconds.

Viewing Eldo Documentation

This release includes the Mentor Graphics Documentation System. The System consists of the InfoHub™ (index), as well as PDF and HTML documents. You access the documentation system using any of the following methods:

- **mgcdocs** shell command from a UNIX or Linux shell window
- **Start** > **Mentor Graphics** > **AMS_<release>** > **documentation** menu from Windows

Command Line Help

A simple online help can also be accessed from the command line with:

```
eldo -help [commands|devices|sources|manual]
```

`commands`	Opens Simulator commands quick help (PDF)
`devices`	Opens Device models quick help (PDF)
`sources`	Opens Sources and Macromodels quick help (PDF)
`manual`	Opens full *Eldo User's Manual* (PDF)

Each of the first three help options will open a link document in Acrobat Reader, which will then allow you to select the command, device model, source or macromodel you require information on.

Entering `eldo -help` without any option will display the list of available topics.

This flag can be specified without the `cir_filename` which is usually mandatory.

Multi-Threading Eldo Simulations

You can activate multi-threading for DC and transient Eldo simulations to speed-up simulation. Eldo will share computer resources on a multi-processor machine. Eldo will make use of all the possible CPUs on the machine.

Eldo will then use these processors as much as possible. It will share the work between the different CPUs in order to speed-up simulation. Note that the CPUs should not already be in use, otherwise simulation will be slower.

Multi-threading is activated with the `-use_proc n | HALF | MAX k | ALL | HYPER` command line argument, which provides you with the flexibility to choose how many of the processors to use. You can specify the number of processors, half the processors, a maximum of k processors, all processors except hyperthreading, or all processors including hyperthreading processors.

Multi-threading can also be activated with the `-mthread` command line argument, or with option `MTHREAD`.

Another alternative activation is with the `-usethread val` command line argument, or option `USETHREAD=VAL`. Eldo will make use of the number of CPUs specified with this flag/option. This is useful if you want to use a smaller number of threads than the number of available CPUs. For example on a 16 CPU machine, you might want to only use six. The number specified can exceed the number of CPUs available, but this is not recommended. If you attempt to set more than twice the number of processors Eldo detects, then it will be capped to twice the number of processors.

Note

`-usethread` takes priority over `-mthread` when both flags/options are specified.

Statistics, generated at the end of simulation, show how many CPUs have been used for the current simulation. This number will also be displayed at the beginning of the TRAN simulation.

The `-cntthread` argument can be specified to provide more details about the computer architecture, for example:

- Bi-Xeon dual core with no hyper-threading

```
Number of physical processors     : 2
+   Hyper-Threading Technology    : disabled
+   Number of cpu cores           : 4
+   Number of logical processors  : 4
```

- Bi-Opteron

```
Number of physical processors     : 2
+   Hyper-Threading Technology    : N/A
+   Number of cpu cores           : N/A
+   Number of logical processors  : N/A
```

- Bi-Xeon (PIV) with hyper-threading

```
Number of physical processors     : 2
+   Hyper-Threading Technology    : enabled
+   Number of cpu cores           : 0
+   Number of logical processors  : 4
```

Eldo will make use of the multi-thread capability of the machine it is running on only if all the following conditions are met:

- the **-USE_PROC** flag has been set, or the **MTHREAD** flag has been set (via option or at invocation)
- the machine is a multi-CPU machine
- the circuit contains devices which are thread-safe
- the circuit contains more than 1000 MOSFET devices per thread (this is not a strict rule, simply a guideline to the design complexity required for multi-threading; it can be used on a circuit that contains less MOSFETs but with many parasitic resistors/capacitors)

If the circuit does not contain any thread-safe device, multi-threading will be ignored. Thread-safe devices are BJT, diode, resistor, capacitor, and MOSFET exclusively. A few models of these categories are not thread-safe, and are therefore handled in a single processor. Eldo will notify when it cannot apply thread optimization on models. Thread-safe and non-thread-safe models can be used in the same circuit.

Eldo automatically adjusts the number of threads so each processor has enough work to do. If Eldo cannot set the number of threads so that each thread has approximately 1000 MOSFETs (or parasitic resistors/capacitors), then multi-threading will be disabled.

When multi-threading is activated, the CPU time reported by Eldo is the sum of the CPU time of all threads. This time will be much higher than the GLOBAL ELAPSED time reported by Eldo. The below examples is an extract of a transcript showing this:

```
# -----     Summary run statistics      ----
#    Global Threads cpu Time 72h 27mn 31s 620ms
#    Global Elapsed Time 47h 36mn 21s
```

Hyper-threading can also be taken into account in conjunction with multi-threading by specifying the `-useht` command line argument. By default, Eldo does not take hyper-threading into account, because it is not always efficient to do so.

Notes

- Multi-threading and DC computation

  DC computation can be very sensitive to numerical noise (because of the algorithm used), and as multi-threading cannot preserve the order of operations, it is possible to obtain different DC computation times and even different DC solutions found from one run to another. This can occur if the circuit has multiple valid DC points, and is the same whether using multi-threading or not.

Linux

When an application is multi-threaded on Linux, the “top” or “ps” operating system commands may return misleading information, depending on the version of the library *libpthread.so* provided with the Linux kernel. Each thread may appear as a separate process, each of them consuming the same memory resources. You should consider that the application only uses the resources indicated for one “process.” Do not consider the addition of all indicated processes, which in reality correspond to the threads.

Eldo Initialization File

An Eldo system initialization file named *eldo.ini* can be created. This file will be interpreted and loaded at the very beginning of each simulation. “`Loading eldo.ini`” is displayed whenever a valid *eldo.ini* file is found. Specifying the `-noinit` argument will disable the loading of the *eldo.ini* file.

This initialization file can be used to specify some configuration options always included in the *.cir* file.

The search order is:

- path specified by environment variable *$ELDO_INI_FILE_PATH*
- current directory
- *$HOME* directory

This file is separated into blocks which are specified using a tag (no mandatory order):

- environment variables definition (after tag `[env]`)
- arguments for command line (after tag `[argu]`); arguments in this section are interpreted before any Eldo command line arguments, Eldo command line arguments have higher priority
- netlist commands (after tag `[eldo]`); commands in this section are interpreted as if they had been included in the netlist with a **`.INCLUDE`** command

A typical *eldo.ini* file may look like:

```
# This line is a comment
[env]
# There must be no blanks between variable name, equal sign
# and variable value
OPTION_DIR=.
MODEL_DIR=../models
LIB_DIR=../libs

[argu]
-outpath $OPTION_DIR/results
-gwl jwdb
-compat

[eldo]
.option noascii notrc
.include $OPTION_DIR/options.inc
.option post probe
```

Location Maps

Location maps are used to replace prefixes of physical pathnames with environment variables (soft pathnames). The location map defines a mapping between physical pathname prefixes and environment variables. IC Flow users can benefit from this functionality to run Eldo directly on a DA-IC generated netlist.

By default, Eldo will search for a location map file in the following locations, in order:

- the filename specified by option **`USE_LOCATION_MAP`** in the *.cir* file, if not found Eldo displays a warning message
- the path stored in the environment variable *$MGC_LOCATION_MAP*

- the netlist directory (*./mgc_location_map*)
- your home directory (*~/mgc_location_map*)

When option **`USE_LOCATION_MAP`** is specified, a warning message is displayed if Eldo does not find a location map file. When option **`USE_LOCATION_MAP`** is not specified, no warning message is displayed by Eldo.

You can disable the load of the location map file by specifying the `-ignore_location_map` command-line argument or specifying option **`IGNORE_LOCATION_MAP`** in the netlist.

Location Map Structure

The purpose of the location map file, in the Eldo context, is to be able to map a soft path with a hard path.

The location map file structure is as follow:

- The first line is a header. That header is not mandatory with eldo which always ignore the first line but is necessary if that file has to be used inside DA-IC. In such a case, please refer to the *DA-IC Design Manager User's Manual* to use the right header.
- The file is then composed of a set of soft pathname / hard pathname and comments.
    - A soft pathname always begins with a dollar sign ($).
    - A hard pathname always begins with the customary slash (/)
    - A comment always begins with sharp (#)

    The user can specify zero, one, or several hard pathname for a soft pathname.

    Here is an example of mgc_location_map file:

```
MGC_LOCATION_MAP_2

#Here is a soft pathname with no hard pathname.
#In such a case, there must be an environment variable named REF_DIR
#and eldo will import the hard pathname contained in that variable
#and will map REF_DIR with that imported hard pathname
$REF_DIR

#Here we have another soft pathname mapped with two hard pathnames
$SOURCE_PROJECT
/home/user/projxxx
/mnt/remote_device/home/user/projxxx
```

Note: when several hard pathnames are given for a soft pathname, only the first one will be used by Eldo. Therefore that pathname must be visible from all computers that are going to use that location map file. Giving several hard pathnames for one soft pathname is only valid in the context of DA-IC. Because, DA-IC can try, given a path, to find the mapped soft pathname. As mount points can be different according to the computer used and the network structure, it might be necessary to specify all mount points of a given folder. But Eldo only tries to expand a soft pathname into a hard pathname to resolve a file name (in a .include statement for example). This is why the first path has to be visible whatever computer is used on the network. More details are available in the *DA-IC Design Manager User's Manual*.

- It is also possible to include files using the INCLUDE command:

```
INCLUDE <hard pathname to the location of the file to include>
```

The included files must not contain a header.

Notes

- When a soft pathname is defined several times in the location map file, only the first definition is used, others are ignored.
- For a given soft pathname, if Eldo is unable to access to the specified hard pathname, a warning message is printed out (warning 939).
- If a given soft pathname already exists as an environment variable, Eldo always uses the value of the environment variable.

eldo-stacktrace File

If an unexpected problem is detected by Eldo (for example, a message that reports an "Error Code"), the file *eldo-stacktrace.dump* or *eldo-stacktrace.vstf* will be generated in the output directory and the simulation will terminate. Include the *eldo-stacktrace* file along with the model causing the error when submitting your Service Request to Mentor support. For more information on Service Requests, please refer to the section Tracking Service Requests, DRs, and ERs of the *AMS Release Notes*.

Statistics File

The statistics output file from Eldo can be useful to understand how a design behaves. It can help to monitor the simulation performance, determine circuit size impact on simulation, debug simulation slowdown, and determine which nodes and blocks should be treated for minimizing rejections. The content of the file is divided into the following main parts:

- General Design Info
- Tool Versions
- High-level Design Information
- Analog Elaboration Information

In the Elaboration and Simulation parts, statistics are grouped into three main categories: Mixed-Signal, Digital, and Analog (SPICE, Fast-SPICE).

Generation

The statistics file can be generated by launching Eldo with the **-stat** flag , see “-stat [-statfile filename]” on page 55.

By default, the output file is named:

```
<cir_filename>_<date>_<time>.stat
```

where *cir_filename* is the name of the *.cir* netlist file to be simulated. For example, **eldo ad4b.cir -stat** executed on June 24 2009 at 9.23am will generate a statistics file named *ad4b_20090624_092325.stat*. To specify a different filename, use the **-statfile** flag, for example, **eldo ad4b.cir -stat -statfile ad4b.stat** will generate a statistics file named *ad4b.stat*.

By default, the output file is generated in the directory where Eldo was launched. Specify the **-outpath** flag to generate the statistics file in a different location.

By default, a single statistics file is set to a maximum size of 50 MB. If the maximum size is reached, a new statistics file is created and a message appears in the transcript. The names of these files are of the format:

```
<cir_filename>_<date>_<time>.<n>
```

where `<n>` = 1, 2, 3, and so on.

Content

Statistics files are divided into sections highlighted by asterisks.

General Design Information

Heading

```
**************************************
*     General Design Info            *
**************************************
```

Content

- **Design name**: Netlist Design Name
- **Machine**: Machine and Platform information
- **Starting time**: Starting simulation time and date record

Tool Versions

Heading

```
*************************************
*     Tool versions                 *
*************************************
```

Content

- Tool versions report

High-Level Design Information

Heading

```
*************************************
*   High-level Design Information   *
*************************************
```

Content

- **Eldo kernel options**: a list of Eldo kernel options used for the simulation

Analog Elaboration

Heading

```
*************************************
*    Elaboration: Analog            *
*************************************
```

Content

- Listing of number of **Nodes** in Eldo/FastSPICE/Both/Total
  - o Number of nodes
  - o Number of device internal nodes
  - o Number of stimulus nodes
- Listing of number of **Devices** in Eldo/FastSPICE/Both/Total
  - o Number of resistors
  - o Number of capacitors
  - o Number of grounded capacitors
  - o Number of inductors
  - o Number of voltage sources

- o Number of current sources
- o Number of controlled sources (E/F/G/H)
- o Number of diodes
- o Number of BJTs
- o Number of JFETs
- o Number of MOS
- o Number of switches
- o Number of transmissions lines
- o Total number of the above listed devices

Memory Used During Elaboration

Heading

```
************************************
*    Elaboration: Memory           *
************************************
```

Content

- Memory used in kB by the Eldo kernel
- Memory allocated and memory used in kB by EZwave and the JWDB server during the elaboration phase
- WDB file name
- Database name with the number of waveforms and the number of aliases

SPICE Elaboration

Heading

```
************************************
*    Elaboration: SPICE            *
************************************
```

Content

This section is dedicated to the elaboration between Eldo and Fast-SPICE (ADiT) if any.

- A table, showing the ratio of the nodes and devices that are on the Eldo side and on the ADiT side.

- A list of the numbers of nodes:
    - **Number of Boundary OUT nodes**: the number of nodes computed by Fast-SPICE (taking care of Eldo current loading of the Node) with values given to Eldo
    - **Number of Boundary IN nodes**: the number of nodes computed by Eldo (taking care of the Fast-SPICE current loading of the Node) with values given to the Fast-SPICE solver
    - **Number of Boundary KEEP nodes**: the KEEP nodes are OUT boundary nodes but declared as A2D nodes. There is no current on those nodes. The voltage values are directly fixed on the digital side.
    - **A2A - ELDO V Control nodes**: the number of voltage sources which are used on the Fast-SPICE side but controlled by Eldo during the simulation
    - **A2A - High Coupling nodes**: the number of highly coupled nodes between the Fast-SPICE and Eldo solvers (analog versus analog)
    - **Number of D2A supply nodes**: the number of D2A boundary elements associated with V source nodes
    - **Number of D2A boundary nodes**: the number of D2A boundary elements inserted between Eldo and Fast-SPICE
    - **Number of A2D boundary nodes**: the number of A2D boundary elements inserted between Eldo and Fast-SPICE

Chapter 3
Eldo Control Language

Introduction

The Eldo *.cir* control file contains all the information necessary to run an Eldo simulation. The Eldo Control Language is used to specify all circuit descriptions and simulation commands in the *.cir* file. The Eldo Control Language is a superset of the standard Berkeley SPICE 2G6 language. Standard Berkeley SPICE control files will thus be accepted by Eldo. However, Eldo provides additional features not available in SPICE. This chapter provides an overview of the *.cir* file structure and general aspects of the language syntax. The following chapters then provide full definitions of the Eldo control language syntax for device, source and macromodel instantiations, and all of the command set.

Overview of the .cir File Structure

Example

As in SPICE, the first line is treated as comment. See "General Aspects of the Language Syntax" on page 70 for descriptions of the syntax.

```
my_example_circuit

* Model definitions
.model m1 nmos level=3 vto=1v       ! Example comment
*                                     more comment
+ uo=550 vmax=2.0e5 cgdo=0.4p

* Subcircuit definitions
.subckt inv 1 2 3
m2 2 1 0 0 m1 w=10u l=4u ad=100p pd=40u as=100p
m1 2 1 3 3 p1 w=15u l=4u ad=100p pd=40u as=100p
c1 2 0 0.5p
.ends inv

* Subcircuit calls
x1 1 2 6 inv
cload 4 0 1p
```

```
* Electrical source definitions
vdd 6 0 5v
vin 1 0 pulse (0 5 10e-9 5e-9 5e-9 30e-9 50e-9)

* Simulation options & commands
.tran 0.5n 100n uic
.ic v(1)=0
.plot tran v(1) v(2) v(3) v(4)
.print tran v(1) v(2) v(3) v(4)
.option eps=0.5e-3 tnom=50 list node
.end
```

General Aspects of the Language Syntax

The following sections in this chapter provide a summary of the Eldo language syntax.

The later chapters in this manual contain the complete descriptions of all the devices, sources, macromodels and commands listed in the following pages. This chapter is designed as a quick reference to the syntax for these and is of use to the more experienced user.

i See "Documentation Conventions" on page 38 for a detailed description of the meanings of the different fonts, brackets, and so on, used throughout this manual.

First Line

The first line is format free and reserved for the circuit title. This line is mandatory and serves as the heading on graphical results output.

Continuation Lines

The length of one input line is limited to 2000 characters. A line may be continued by using the `+` character at the beginning of the new line. In arithmetic expressions, this leading `+` sign will be ignored arithmetically and treated as a continuation. Two `+` signs may be placed together to both continue the line, *and* perform the addition operation.

Comment Lines

A new comment line must begin with the `*` character. If the comment follows an Eldo statement on the same line (inline comment), it must begin with the `!` (or, in some cases, the `*`) character. The inline comment character (`!` or `*`) must be preceded by a white space. Otherwise, the `!` or `*` character will be considered as a valid character that can, for example, be used in node names, or in other cases be ignored. The `*` character not allowed as an inline comment on **.EXTRACT**, **.DEFMAC**, **.DEFWAVE**, **.MEAS**, **.OBJECTIVE**, and **.PARAM** commands.

```
* <comment line>
<Eldo statement> ! <inline comment>
```

Note

The inline comment character (`!` or `*`) must be preceded by a white space. If not, Eldo could ignore it in the case of a parameter value, or consider it a valid character in node names. For example, the first of the below lines is accepted, without any error or warning message, to be identical to the second ("5!" is interpreted as the number 5 and the 22 is parsed as the TC1 temp coefficient).

```
r1 1 2 5! 22
r2 2 0 5 22
```

A set of comment lines can also be grouped together into a block as shown below:

```
#com
A block of
comment
#endcom
```

In these cases the `*` character is not needed.

Component Names

Component names start with the component reserved characters and continue with an arbitrary sequence of alphanumeric characters, including `%`, `$`, `#`, _. Component names cannot be broken at the end of a line.

Parameter Names

Parameter names may contain an arbitrary sequence of alphanumeric characters, including `%`, `$`, `#`, _, `!`. Parameter names cannot be broken at the end of a line. Parameter names should not contain boolean operators. Such a name can be quite ambiguous.

String Parameters

Eldo accepts quoted character strings as parameter values. These string values may be used for model names and filenames. To use a string as a parameter, enclose the string within double quotes, for example:

```
.param TT1="ResMod"
```

To maintain the case of the string enclose the string within double quotes first and then enclose within single quotes, for example:

```
.param TT2='"PwlModFile.src"'
```

A string parameter can be defined on multiple lines if the string on each line is enclosed within double quotes, for example:

```
.SIGBUS mybus base=bin TFALL=3n TRISE=3n THOLD=20n TDELAY=50n VHI=5 VLO=0
+ pattern $ (PAT)
.PARAM PAT= " 00000 00001 00010 00011 00100 00101 00110 00111 "
+ " 01000 01001 01010 01011 01100 01101 01110 01111 10000 10001 "
+ " 10010 10011 10100 10101 10110 10111 11000 11001 11010 11011 "
+ " 11100 11101 11110 11111 00000 00001 00010 00011 00100 00101 "
```

The space before the double quote at the right hand side is mandatory. Without it, Eldo will generate an error.

The value of a string is retrieved simply by specifying the dollar sign ($) and parentheses ( ). Examples:

```
.param MOD="Pmos1"
m1 d g s b $(MOD) w=1u l=1u

.param STIMFILE='"Stim.txt"'
v1 1 0 pwl file=$(STIMFILE)  R
```

String parameters can be used in **.SIGBUS** or pattern sources. A sweep (for example **.STEP**) can also be made on string parameters.

Reserved Keywords

Some keywords should not be specified in a **.PARAM** command. If they are then errors will be generated. See Table 10-21 and Table 10-22 on page 779.

Node Name Conventions

Node names may contain an arbitrary sequence of alphanumeric characters and some non-alphanumeric characters including: !, $, #, _, [], <>, :, \, /, |, +, -, *, %.

The following characters are not allowed in node names: (), {}, ', =. Node names can not begin with any of the following characters: *, +, -, &, |, ". Do not use the period character (.) in a node name because it is reserved as a hierarchy separator between a subcircuit name and a node name.

Node names cannot be broken at the end of a line. If the first character of a node name is numeric then it is forbidden for an alphabetic character to follow in the same name: all characters must then be numeric. Numeric characters can, however, follow an alphabetic character in the same node name.

1TOTO	Illegal node name
123	Legal node name
TOTO1	Legal node name

Whenever Eldo sees a node name with suffix ".0", Eldo assumes that node 0 was intended if the name also begins with an X, for example: `C1 a XA.0 1p` is identical to `C1 a 0 1p`. Alternatively, with syntax: `C1 a a.0 1p`; a node named `a.0` is created as a regular node.

Node Names Used Inside Subcircuits

If you wish to access nodes from a higher level of hierarchy than that in which they are defined, it may be done as shown in the following example:

`X27.X113.N3` Legal node name

The node `N3` is located within a subcircuit `X113` which, in turn, is located inside another subcircuit `X27`.

For more information about the usage of nodes inside subcircuits, please refer to ".SUBCKT" on page 898.

Values

Values are always handled as real numbers. They may be specified in exponential notation or with scale factors.

Model Names

Model names cannot start with a numeric. This causes compilation of the netlist to be broken, giving an error message.

Scale Factors

For scaling, you can choose between the exponential notation, or one of the following:

Table 3-1. Scale Factors

Symbol	Multiplier	Name
A	1.0×10^{-18}	atto
F	1.0×10^{-15}	femto
P	1.0×10^{-12}	pico
N	1.0×10^{-9}	nano
U	1.0×10^{-6}	micro
M	1.0×10^{-3}	milli
K	1.0×10^{3}	kilo

Table 3-1. Scale Factors

Symbol	Multiplier	Name
MEG	1.0×10^6	Mega
G	1.0×10^9	Giga
T	1.0×10^{12}	Tera

Notes

- Letters which are not scale factors are ignored if they immediately follow a number. Hence 10, 10V and 10Hz all represent the same number, 10. However, 10A will be interpreted as $1.0e^{-17}$, because of the atto scaling factor.
- Letters immediately following a scale factor are ignored. Thus M, MA, MSEC, and MMHOS all represent the same scale factor, M.
- The scale factor `M` represents 1×10^{-3} or “milli” units. 1×10^6 or “mega” units are specified using the `MEG` scale factor. This is commonly confused in SPICE syntax.
- Scale factors are not cumulative. `KK` is not `MEG`, but `K`, since the second letter is ignored.
- M.K.S. units are used throughout the netlist.

Directives

IF/ELSE/ELSEIF/ENDIF Condition Statements

IF-ELSE-ELSEIF-ENDIF condition statements can be used inside the Eldo netlist, and can be used to instantiate devices depending on parameter size. ELIF and ELSIF keywords are also accepted as the ELSEIF condition.

- The IF-ELSE-ELSEIF-ENDIF section must not contain any command, otherwise unexpected results can be obtained. Warnings are issued in this case.
- The parameters used in expressions of if statements are evaluated during the parsing of the design only, even if these parameters are changed at run time (because of **`.STEP`**, for instance), then the design will not be changed, a message will be issued that the operation cannot be taken into account. The **`.ALTER`** mechanism should be preferred here.

Example of usage:

```
.subckt foo 1 param: p1 = 1
if (p1 >= 1.5)
r1 1 0 '2*p1'
else
r1 1 0 '3*p1'
```

```
endif
.ends

i1 1 0 3
x1 1 foo p1 = 3
.op
.end
```

In this example, r1 value will be 6 Ω.

Note

Errors are issued if ADMS is used, or if X or Y statements are present inside if-endif blocks.

String Operator on If Expressions

It is possible to use C-like functions **strcmp** and **strncmp** inside **if** statements used for netlist configuration. **strcmp** and **strncmp** return "0" when the string matches, "not 0" otherwise. Examples:

```
.param p1 = "myp1"

if (strcmp(p1,"myp1") == 0)
i1 1 0 1
else
i1 1 0 2
endif
r1 1 0 1
```

Here, i1 will be 1A.

```
if (strcmp($(p1),"myp1") == 0)
i1 2 0 3
else
i1 2 0 4
endif
r1 2 0 1
```

Similarly, i1 will be 3A because the **if** statement is true.

Directives Interpreted by the Eldo Parser (Default)

By default, (without the **-E** or **-EE** flag) Eldo understands the following simple pre-processor commands: #if, #define <name>, #ifdef <name>, #ifndef <name> #else, #endif.

These can be used to select a part of the netlist, for example:

```
#define NO_RESISTOR
...
#ifndef NO_RESISTOR
R1 A B 1k
```

```
#endif
...
```

A *define* can also be specified at invoke time with:

```
eldo <arguments> -define <name>
```

For example, **-define** *foo* on the command line is equivalent to #define foo in the netlist. For both methods, the netlist statement #ifdef foo will be true.

The #if statement can be used for making complex conditional statements using logical operators: OR ||, AND &&; and comparison operators: not equal to !=, equal to ==. The #if directive can also be used in conjunction with the defined function, that is, if a #define <name> is active, defined(<name>) returns 1 whatever value is assigned to <name>.

Example

```
V1 1 0 1
#define U 0
#if (defined(U) || defined(A))
R1 1 0 1
#else
R1 1 0 2
#endif
.end
```

Because U is defined in the above example, defined(U) will return 1. The #if statement is true and R1 will be set to 1. When the #if statement is substituted with: #if (defined(U) && defined(A)), the #if statement will no longer be true because A is not defined. R1 would therefore be set to 2.

#NETLIST_END and #END_NETLIST_END Directives

Syntax:

```
#NETLIST_END
...
#END_NETLIST_END
```

Every line declared between these two directives is postponed until the end of the netlist (**`.END`**). This could be used, for example, to allow a **`.ALTER`** definition anywhere in the netlist.

If several #NETLIST_END/#END_NETLIST_END blocks are declared, they are treated according to their order in the design. These blocks can be defined in the top netlist or any included files. Nested #NETLIST_END/#END_NETLIST_END directives are ignored.

Example

```
i1 1 0 dc 'p1'
.dc
```

```
#netlist_end
.alter
.param p1=3m
#end_netlist_end

.param p1=1m
r1 1 0 1
.extract dc v(1)
.alter
.param p1=2m
.end
```

In this example, three DC analyses will be performed: the first one using p1=1m, the second one using p1=2m, and the last one using p1=3m which is defined in a #netlist_end/#end_netlist_end block.

Directives Interpreted using the C Pre-Processor (-E/-EE Arguments)

For an extended use of pre-processor commands to define macros and replace them inside the netlist (see the example below), the **-E** or **-EE** flag needs to be specified. These are not the Eldo default because the parsing of large circuits may be significantly slower.

The **-E** flag forces Eldo to provide only the main netlist to the C pre-processor, whereas **-EE** ensures that all `#include filename` statements will be pre-processed before parsing. The **`.INCLUDE`**/**`.LIB`** statements, which are SPICE commands, are not understood by the C pre-processor and so will not be pre-processed.

The #include directive can only be used when the **-E**/**-EE** flags are specified.

Note: The C pre-processor analyses files independently. Therefore #define statements are only known to the file they are defined in.

The **-define** flag can also be specified on the command line, and it will have the same effect as without the **-E**/**-EE** flags. To use #define in Eldo in the same way you define macros in the C language, the syntax is as follows:

```
#define macro(args) expression
```

When the macro is encountered in the netlist, it is replaced by the expression. Arguments of the macro will be replaced by the literals in the macro call. For example:

```
#define sat_margin(device) abs(vds(device)-vdss(device))
.extract sat_margin(XM0.M1)
```

will be replaced by:

```
.extract abs(vds(XM0.M1)-vdss(XM0.M1))
```

Note

Because **-E**/**-EE** uses the C pre-processor, you must ensure that there is a carriage return proceeding the directive in the file to avoid problems. Uppercase function names are not accepted. Using comments defined with #com and #endcom are not compatible with **-E**/**-EE**.

Arithmetic Functions

A set of arithmetic functions may be used in Eldo for the calculation of device parameters, model parameters, new waves, and so on. These are listed below:

Table 3-2. Arithmetic Functions and Operators

Function	Returns
SQRT(VAL**)**	Square root of VAL
LOG(VAL**)**	Neperian logarithm of VAL
LOG10(VAL**)**	Decimal logarithm of VAL
DB(VAL**)**	Value in dBs of VAL ($20 \times \log_{10}$(VAL))
EXP(VAL**)**	Exponent of VAL
COS(VAL**)**	Cosine of VAL, where VAL is defined in radians
SIN(VAL**)**	Sine of VAL, where VAL is defined in radians
TAN(VAL**)**	Tangent of VAL, where VAL is defined in radians
ACOS(VAL**)**	Arc cosine of VAL
ASIN(VAL**)**	Arc sine of VAL
ATAN(VAL**)**	Arc tangent of VAL
COSH(VAL**)**	Hyperbolic cosine of VAL
SINH(VAL**)**	Hyperbolic sine of VAL
TANH(VAL**)**	Hyperbolic tangent of VAL
SGN(VAL**)**	Returns the signum of VAL: +1 if VAL>0, 0 if VAL=0, -1 if VAL<0
SIGN(VAL**)**	Returns the signum of VAL: +1 if VAL>=0, -1 if VAL<0
SIGN(VAL1, VAL2**)**	Returns **ABS(**VAL1**)**×**SGN(**VAL2**)**
PWR(VAL1, VAL2**)**	Returns the absolute value of VAL1, raised to the power of VAL2, with the sign of VAL1
POW(VAL1, VAL2**)**	Returns the value of VAL1 to the power of the integer part of VAL2
ABS(VAL**)**	Absolute value of VAL: |VAL|
INT(VAL**)**	Integer value of VAL (equivalent to **TRUNC**)

Table 3-2. Arithmetic Functions and Operators

Function	Returns
TRUNC(VAL**)**	Truncated value of `VAL` (Integer part of real value)
ROUND(VAL**)**	Rounded, to the nearest integer, value of `VAL`
CEIL(VAL**)**	Ceiling rounding function, returns the smallest integer value not less than `VAL`. This is known as rounding up. For example ceil(1.25) returns 2.0 and ceil(-1.25) return -1.0.
FLOOR(VAL**)**	Floor rounding function, returns the largest integer value not greater than `VAL`. This is known as rounding down. For example floor(1.25) returns 1.0 and floor(-1.25) return -2.0.
MIN(VAL1,..., VALn**)** **DMIN(**VAL1,...,VALn**)**	Returns the minimum of `VAL1` to `VALn`. There is no limit to the number of values that can be specified
MAX(VAL1,..., VALn**)** **DMAX(**VAL1,...,VALn**)**	Returns the maximum of `VAL1` to `VALn`. There is no limit to the number of values that can be specified
DERIV(VAL**)**	Returns the derivative of `VAL`
REAL()	Returns the real part of a complex number
IMAG()	Returns the imaginary part of a complex number
MAGNITUDE()	Returns the magnitude of a complex number
CONJ()	Returns the conjugate of a complex number
COMPLEX(a, b**)**	Returns a complex number using ‘a’ as the real part and ‘b’ as the imaginary part
STOSMITH(val**)** **YTOSMITH(**val**)** **ZTOSMITH(**val**)**	Returns a normalized value of a complex quantity. These functions can be used to convert complex functions (S/Y/Z parameters generated by a **`.STEP`** analysis) so they can be correctly plotted in a Smith chart; an example is given in the .PLOT section of the Simulator Commands chapter.
DDT(VAL**)**	Returns the derivative of `VAL`
IDT(VAL**)**	Returns the integral of `VAL`
LIMIT(a, b, c**)**	Returns `b` if `a` < `b`, returns `c` if `a` > `c`, returns `a` otherwise
BITOF(a, b**)**	Returns “1” if bit `b` of the integer value of parameter `a` is a “1”. Returns “0” if bit `b` of the integer value of parameter `a` is a “0”
PWL(xvalue, interp, x1, y1, ... xn, yn**)**	Returns the equivalent output value at the input value `xvalue`, `interp=0|1` specifies whether the y value is interpolated linearly (1) or not (0). `xn` and `yn` are used to calculate the equivalent output value

Table 3-2. Arithmetic Functions and Operators

Function	Returns
PWL_CTE(xvalue, interp, x1, y1, ... xn, yn**)**	Returns the equivalent output value at the input value `xvalue`, `interp=0|1` specifies whether the y value is interpolated linearly (1) or not (0). `xn` and `yn` are used to calculate the equivalent output value. This function plots all specified points but will hold the first specified value if the simulation start time is less than the first time point specified in the function. Similarly it will hold the last value until the simulation is complete.
PWL_LIN(xvalue, interp, x1, y1, ... xn, yn**)**	Returns the equivalent output value at the input value `xvalue`, `interp=0|1` specifies whether the y value is interpolated linearly (1) or not (0). `xn` and `yn` are used to calculate the equivalent output value. This function linearly extrapolates the first value using the first two points of the function. The last value will also be linearly extrapolated using the last two points. This will only happen if the simulation start time is less than the first time value specified in the function, and the simulation end time is greater than the last time value of the function.
ONGRID([offset], step, value**)**	Returns 1 if `value` = `offset` + k×`step` where k is an integer value. Otherwise it returns 0. Default value of offset is 0.0.
SELECT(table_name, xvalue [, LINEAR | SAMPLE_HOLD |SPLINE])	Returns a value interpolated from an array of (x,y) values according to a given x value. table_name is the name of an array of (x,y) values defined with a **`.TABLE`** command, xvalue is the x value for which y is requested, LINEAR | SAMPLE_HOLD | SPLINE indicates the interpolation method to use to calculate the y value. The default is SAMPLE_HOLD.
SMMIN(a, b, eps**)**	Returns the minimum of a and b, with smoothing coefficient eps
SMMAX(a, b, eps**)**	Returns the maximum of a and b, with smoothing coefficient eps
SMABS(val, eps**)**	Returns the absolute value of val, with smoothing coefficient eps
SMSGN(val, eps**)**	Returns the signum of val, with smoothing coefficient eps
SMSIGN(val, eps**)**	Returns the signum of val, with smoothing coefficient eps
SIGMA()	Returns the sum of various items (numbers, parameters, or output quantities); wildcards are accepted for output quantities
MOD(x,y**)**	Modulo operator. Floating-point remainder value function of dividing `x` by `y`. Returns the value $x - i \times y$, where *i* is the quotient of `x / y`, rounded towards zero to an integer. An error is returned if `y` is 0.

Notes

- Where `VAL` is written above, it normally means a numeric value, but in certain cases, the functions may also be applied to waves.

- The **MAX** and **MIN** functions are automatically converted into **DMAX** and **DMIN** functions when necessary, for example:
  `R1 1 2 {min(2,1,5)}` is equivalent to:
  `R1 1 2 {dmin(2,1,5)}` which is also equivalent to:
  `R1 1 2 1`
  There is no limit to the number of arguments that can be used in **MAX**, **MIN**, **DMAX** and **DMIN** arguments.
- If three arguments are specified for **MIN** (that is, **MIN**(a,b,c)), this represents the **MIN** of waveform a in the time window [b,c]. Use **DMIN** to return the minimum of a, b and c as computed at each time step.
- The **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** functions, when used in bias-dependent expression (such as in R(V)), can cause non-convergence due to these operators making the function non-derivable. Use option "MMSMOOTH" on page 980 to make them derivable, to avoid such problems. Alternatively use the **SMMIN**, **SMMAX**, **SMABS**, **SMSGN** and **SMSIGN** operators.
- The functions **REAL**, **IMAG**, **MAGNITUDE**, **CONJ**, and **COMPLEX** can only be used in **.EXTRACT** and **.DEFWAVE** expressions.
- **DDT(**VAL**)** and **IDT(**VAL**)** can *only* be used on E & G elements, see the corresponding descriptions for the "Voltage Controlled Voltage Source" on page 350 and "Voltage Controlled Current Source" on page 363. Expressions associated with R/L/C devices do not support DDT/IDT operators.
  The **DDT** operator differs from the **DERIV** operator in the sense that **DDT** utilizes the integration scheme used by Eldo, while the **DERIV** operator exclusively uses the Backward-Euler algorithm.
- There can be differences between the **DERIV** operator of Eldo and the equivalent operator (**DRV**) of EZwave. The formula used for the Eldo **DERIV** operator is Backward-Euler (that is, deriv(f(t)) = (f(t) - f(t-h))/h) with *t* being the current time and *h* being the time step, however the **DRV** operator of EZwave uses a more complex formula based on f(t), f(t-h) and f(t+h). Eldo cannot use the EZwave "post-processor" formulation, because it requires to know the value at *t+h*, which is not possible when the simulator is running (Eldo at time *t* cannot know the value at *t+h*), but post-processor formulation is usually smoother, hence both formulations are used.
- Nested **COMPLEX()** statements are forbidden, as well as using complex quantities for the real or imaginary part. Corresponding errors are:

```
ERROR 3040: Nested complex(,) functions is not allowed.
ERROR 3041: Complex quantities can not be used inside complex(,) function.
```

- If **CEIL**, **ROUND**, **FLOOR** cause convergence problems, Eldo will generate a warning and invite the user to specify option **NOLTEDISC** to bypass LTE checks.

-compat Flag

When the `-compat` flag is active, the following arithmetic function/operator rules apply:

- log(x) = sign(x) × log(abs(x))
- log10(x) = sign(x) × log10(abs(x))
- db(x) = sign(x) × 20.0×log10×abs(x))
- sqrt(x) is -sqrt(abs(x)) if x is negative.
- x**y is computed as x^y if x is positive, $-(\text{abs}(x)^y)$ if x is negative, and 0 if x is 0.
- The power operator (^) has highest precedence (same as standard Eldo); prior to v6.3_2 it had lower precedence in -compat mode than the multiplication and division operators.

Note
In Eldo standard mode: sqrt(x) returns an error if x is negative; x**y is computed as exp(y×log(x)) if x is strictly positive, 0 if x is 0, and returns an error if x is negative.

Operators

Operator Precedence

The order of precedence and associativity of operators in Eldo affect the evaluation of expressions. For example, in the expression a=2+b*3, which happens first? the addition or the multiplication? Expressions with higher-precedence operators are evaluated first.

Table 3-3 summarizes the precedence and associativity (the order in which the operands are evaluated) of Eldo operators, listing them in order of precedence from highest to lowest. Where several operators appear together, they have equal precedence and are evaluated according to their associativity.

Table 3-3. Operator Precedence

Operator	Description	Associativity
()	function call	left-to-right
! -	logical NOT, unary negation	right-to-left
** ^	power, power (synonym)	left-to-right
* /	multiply, divide	left-to-right
+ -	add, subtract	left-to-right
<< >>	bitwise left shift, bitwise right shift	left-to-right

Table 3-3. Operator Precedence

Operator	Description	Associativity
< <= > >=	less than, less than or equal, greater than, greater than or equal	left-to-right
== !=	equal, not equal	left-to-right
&	bitwise AND	left-to-right
|	bitwise OR	left-to-right
&&	logical AND	left-to-right
||	logical OR	left-to-right

Arithmetic Operators

The arithmetic operators available are **+**, **-**, *****, **/** and **^** (or ******) for power.

Note

The power operator (**^**) has the highest precedence. For example:
1+4^2 gives the result: 17
2*3^2 gives the result: 18

x^y or x**y is computed as exp(y*log(x)) if x is strictly positive, 0 if x=0, and returns an error if x is negative. Specify option **POWNEG0** to allow a negative x value in power expressions: the expression will return 0. In versions of Eldo prior to v6.6, a negative x value was allowed.

Boolean Operators

The following boolean expressions/operators are available:

Table 3-4. Boolean Operators

Operator	Meaning
!=	not equal to
==	equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
||	OR operator
&&	AND operator

Bitwise Operators

The following Bitwise operators are available:

Table 3-5. Bitwise Operators

Operator	Meaning
&	Bitwise AND operator
|	Bitwise OR operator
<<	Bitwise shift left operator
>>	Bitwise shift right operator

Expressions

Expressions can be used in a netlist with certain restrictions.

Numerical expressions must be contained within braces, **{** and **}**, single quotes, **'** and **'**, or parentheses, **(** and **)**.

String expressions should be contained in double quotes, **"** and **"**.

Mathematical grouping within expressions must be done using normal brackets, **(** and **)**.

Constants and parameters may be used in expressions, together with the built-in functions and operators described above.

Expressions may be used in the following situations:

- Parameters in the calculation of MOS geometries and R, C and L values.
- Parameter values in the **.MODEL** command.
- Time point values in the signal descriptions **PULSE**, **PWL**, **SFFM**, **SIN** and **EXP**.
- Parameters values in the **.SIGBUS** command.
- Voltage and current source values.
- S and Z transform (**FNS** and **FNZ**) devices.
- **.PARAM**, **.EXTRACT** and **.DEFWAVE** commands.
- **E** and **G** sources described by functions or tables.
- **R**, **C** and **L** devices described by functions.

Some parameters may appear in expressions but will cause an error if used in a `.PARAM` command. For more information on these see "Reserved Keywords" on page 72.

Examples

```
r1 1 2 {3.0*p1-4k}
.model nn nmos vt0={p2-p2/2.0}
e1 1 2 value={15v*sqrt(v(3,2))}
.defwave pow=v(a)*i(b)
.param x1={2*sqrt(a)}
```

-compat flag

In `-compat` mode, double quotes are considered as single quotes. (In standard Eldo mode, double quotes are used to specify a parameter string.)

Conditional Evaluation of Expressions

Parameters or source values can be evaluated in expressions containing conditional statements.

Syntax

```
VALIF(CONDITION,expression1,expression2)
EVAL(CONDITION?expression1:expression2)
```

If `CONDITION` is TRUE, then **`VALIF`** (or **`EVAL`**) returns `expression1` else it returns `expression2`. The keyword **`VALIF`** (or **`EVAL`**) can be used in any expression. The **`VALIF`** operator also accepts strings in **`.PARAM`** statements. This is useful for selecting models according to parameter values.

Examples

```
.param p1 = 1.0
.param p2 = 2.0
.param p3 = valif(p1>p2,p1+1.5,p2+1.5)
```

Here, `P3` will be assigned the value 3.5.

```
.param p1 =1
.param p2 = 2
.param pu = valif(p1 > p2,"r1","r2")

.model r1 r r = 1
.model r2 r r = 2
i1 1 0 1
r1 1 0 $(pu) 1
```

This examples shows the use of strings. Here, model r2 will be used because condition p1 > p2 is false.

Note

The **EVAL** syntax is closer to 'C' language, and may be more convenient for some users.

Simulation Counters

After a simulation has been completed, Eldo writes simulation information, in tabular form, to the ASCII output (*.chi*) file. The following information is output:

Node and Element Information

Information concerning circuit nodes and elements is written to the *.chi* file in the following format:

```
NUNODS  NCNODS  NUMNOD  NUMEL  DIODES  BJT  JFET  MOSFET
16      16      16      22     0       0    0     19
```

where the parameters have the following definitions:

NUNODS	Number of nodes before subcircuit expansion. Corresponds to the number of top nodes, intrinsic nodes of devices not included.
NCNODS	Number of nodes after subcircuit expansion.
NUMNOD	Total number of nodes including those created by parasitic resistances.
NUMEL	Total number of elements contained in the circuit.
DIODES	Number of diode elements contained in the circuit.
BJT	Number of BJT elements contained in the circuit.
JFET	Number of JFET elements contained in the circuit.
MOSFET	Number of MOSFET transistor elements contained in the circuit.

Grounded Capacitors Information

Information concerning grounded capacitors is written to the *.chi* file in the following format:

```
NUMGC
1
```

where the parameters have the following definitions:

NUMGC	Number of grounded capacitors not taken into account by NUMEL

Matrix Information

When Eldo creates a matrix, the following information is written to the *.chi* file:

```
NSTOP         NTERM            PERSPA
   13           122        7.219e+01
```

where the parameters have the following definitions:

NSTOP	Number of lines in the matrix
NTERM	Number of terms in the matrix
PERSPA	Sparsity coefficient in percent (%)

Newton Block Information

When Eldo creates a number of Newton blocks, the following information is written to the *.chi* file:

```
NBLOCKS    NODEBLK    MAXSIZE    MINSIZE
```

where the parameters have the following definitions:

NBLOCKS	Total number of Newton blocks created
NODEBLK	Number of nodes contained in each Newton block
MAXSIZE	Size of the biggest Newton block
MINSIZE	Size of the smallest Newton block

Convergence Information

Information concerning circuit nodes and elements is written to the *.chi* file in the following format:

```
NUMTTP   NUMRTP  LTERTP  INWCALL  ITERNW     MEMSIZE
80       15      2       243      2.000e+00  581896
```

where the parameters have the following definitions:

NUMTTP	Number of steps accepted by the simulator and sent to the binary output (*.wdb*) file
NUMRTP	Number of steps rejected due to the truncation error being too large
LTERTP	Number of time steps rejected due to LTE
INWCALL	Total number of iterations or Newton calls needed to solve the Newton blocks

ITERNW	Total number of Newton calls for **.OP**, **.DC** and **.AC** analyses and is the average number of Newton calls needed to achieve convergence for a **.TRAN** analysis
MEMSIZE	Memory size allocated to the circuit by Eldo

```
NDEVCALL    NKIRCH     NMAXCALL        ITERM        LATENCY
16038       0          9               1.00e+00     5.208e+00%
```

where the parameters have the following definitions:

NDEVCALL	Number of device calls
NKIRCH	Number of calls or iterations needed to solve Kirchoff’s Law (OSR only)
NMAXCALL	Maximum number of calls needed to solve a time or DC point
ITERM	Average number of OSR loops
LATENCY	Percentage of latency in the circuit

Temperature Handling

Eldo allows temperature handling using the commands **.TEMP**, **TNOM**, **TMOD** and **T** and allows formulation of temperature dependent functions using the variable **TEMPER** (or **TEMP**). These commands and functions are briefly described below:

The **TNOM** function from the **.OPTION** command is used to set the nominal simulation temperature, that is, the temperature at which parameter calculations are made. Default is 27 °C.

Note

TNOM may appear in expressions.
TNOM is a reserved keyword, however it may be specified as a parameter in a **.PARAM** command when **.OPTION DEFPTNOM** is set. The temperature value used by the Eldo model evaluator is always that which is set with **.OPTION** TNOM=val. See “DEFPTNOM” on page 976 and “TNOM=VAL” on page 983.

The **.TEMP** command is used to execute several successive simulations at various temperatures. See “.TEMP” on page 907.

The **TMOD** parameter (in certain models) is used to set the model temperature. The value of this parameter overrides the **.TEMP** command above. The **T** parameter (in certain devices) is used to set the temperature of an individual instance of a device or model. This parameter overrides the **TMOD** command above.

Please refer to the Device Models chapter.

To summarize, the order of priority of the above temperature related commands and parameters is **T**, then **TMOD** and then **.TEMP**, with decreasing priority. That is, **T** has the highest priority.

TEMPER is a variable returned by the simulator which gives the value of the current simulation temperature and may be used in subsequent calculations. This variable will be the present simulation temperature resulting from either a **.TEMP** command, a **.DC TEMP** sweep or, if neither are specified, the value of **TNOM** given in the **.OPTION** command. The **TEMPER** variable may be used in the formulation of temperature dependent expressions. Any expressions containing the **TEMPER** variable will be automatically reevaluated in the case of a change in this temperature.

Note

The **TEMP** variable is synonymous with the **TEMPER** variable. Both refer to the temperature of the circuit.

Example

The **TEMPER** variable may be used in conjunction with **VALUE={**EXPR**}** in resistors, capacitors and inductors to specify devices whose values vary with temperature.

Refer to these components in the Device Models chapter.

```
Cvariable 3 7 VALUE={C0*(1+0.002*(TEMPER^2))}
```

This specifies a capacitor **C**variable connected between nodes 3 and 7 and its value defined as the nominal capacitance C0 multiplied by (1+ 0.002 multiplied by the square of the current simulation temperature **TEMPER**). The **TEMPER** variable may also be used in expressions for model parameters.

Devices

Resistor

```
Rxx N1 N2 [MOD[EL]=MNAME] [VAL] [[TC1=]T1] [[TC2=]T2] [[TC3=]T3]
+ [AC=VAL|{EXPR}] [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [L=VAL] [W=VAL]
+ [STATISTICAL=0|1] [KEEPRMIN] [NONOISE] [KF=VAL] [AF=VAL] [WEEXP=VAL]
+ [LEEXP=VAL] [FEXP=VAL] [FMIN=VAL] [FMAX=VAL] [NBF=VAL]
Rxx N1 N2 [MOD[EL]=MNAME] VALUE={EXPR} [ACDERFUNC=0|1]
+ [RESTORE_CAUSALITY=0|1] [[TC1=]T1] [[TC2=]T2] [[TC3=]T3] [AC=VAL]
+ [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [STATISTICAL=0|1] [KEEPRMIN] [NONOISE]
```

+ [**KF**=VAL] [**AF**=VAL] [**WEEXP**=VAL] [**LEEXP**=VAL] [**FEXP**=VAL] [**FMIN**=VAL]
+ [**FMAX**=VAL] [**NBF**=VAL] [**FIT**=0|1] [**CFMAX**=VAL] [**CDELF**=VAL]
Rxx N1 N2 [[**TC1**=]T1] [[**TC2**=]T2] [[**TC3**=]T3] [**AC**=VAL] [**T**[**EMP**]=VAL]
+ [**DTEMP**=VAL] [**M**=VAL] [**KF**=VAL] [**AF**=VAL] [**WEEXP**=VAL] [**LEEXP**=VAL]
+ [**FEXP**=VAL] [**FMIN**=VAL] [**FMAX**=VAL] [**NBF**=VAL] [**STATISTICAL**=0|1]
+ **TABLE** EXPR [**KEEPRMIN**] [**NONOISE**]
Rxx NP NN **POLY** VAL {COEF} [**TC1**=T1] [**TC2**=T2] [**TC3**=T3] [**STATISTICAL**=0|1]

Capacitor

Cxx NP NN [**MOD**[**EL**]=MNAME] [**DCCUT**] [VAL] [**M**=VAL] [**L**=VAL] [**W**=VAL]
+ [**T**[**EMP**]=VAL] [**DTEMP**=VAL] [**TC1**=T1] [**TC2**=T2] [**TC3**=T3] [**IC**=VAL]
+ [**STATISTICAL**=0|1]
Cxx NP NN **POLY** VAL **{**COEF**}** [**TC1**=T1] [**TC2**=T2] [**TC3**=T3] [**M**=VAL]
+ [**CTYPE**=VAL] [**IC**=VAL] [**STATISTICAL**=0|1]
Cxx NP NN [**VALUE**=]**{**EXPR**}** [**ACDERFUNC**=0|1] [**RESTORE_CAUSALITY**=0|1]
+ [**TC1**=T1] [**TC2**=T2] [**TC3**=T3] [**CTYPE**=VAL] [**STATISTICAL**=0|1]

Inductor

Lxx NP NN [**MOD**[**EL**]=MNAME] [**DCFEED**] [VAL] [**M**=VAL1] [**T**[**EMP**]=VAL] [**DTEMP**=VAL]
+ [**IC**=VAL3] [**TC1**=T1] [**TC2**=T2] [**TC3**=T3] [**R**=VAL4] [**STATISTICAL**=0|1]
Lxx NP NN **POLY** VAL **{**LN**}** [**IC**=VAL] [**R**=VAL] [**TC1**=T1] [**TC2**=T2] [**TC3**=T3]
+ [**STATISTICAL**=0|1]
Lxx NP NN [**VALUE**=]**{**EXPR**}** [**ACDERFUNC**=0|1] [**RESTORE_CAUSALITY**=0|1]
+ [**R**=VAL|**R VALUE**=EXPR|**R**
+ **TABLE** {fval rval}] [**TC1**=T1] [**TC2**=T2] [**TC3**=T3] [**STATISTICAL**=0|1]
Lxx {port_list} **KMATRIX**=data_block [**STATISTICAL**=0|1]
Lxx {port_list} **RELUCTANCE**=({rn,cn,valn}) <options> [**STATISTICAL**=0|1]
Lxx {port_list} **RELUCTANCE** {**FILE**="file"} <options> [**STATISTICAL**=0|1]

Coupled Inductor

Kxx **L**yy **L**zz KVAL [**KR**=KRVAL]

RC Wire

Rxx N1 N2 MNAME [[**R**=]VAL] [**TC1**=VAL] [**TC2**=VAL] [**C**=VAL] [**CRATIO**=VAL]
+ [**L**=VAL] [**W**=VAL] [**M**=VAL] [**T**[**EMP**]=VAL] [**DTEMP**=VAL] [**SCALE**=VAL]
+ [**STATISTICAL**=0|1]

Diffusion Resistor

Pxx N1 N2 [NS] MNAME [**L**=VAL] [**W**=VAL] [**NB**=VAL]

Semiconductor Resistor

Pxx N1 N2 NS MNAME [**R**=VAL] [**L**=VAL] [**CL**=VAL] [**W**=VAL] [**CW**=VAL] [**AREA**=VAL]
+ [**STATISTICAL**=0|1]

Transmission Line

Txx NAP NAN NBP NBN [**Z0**=VAL1] **TD**=VAL2 [**STATISTICAL**=0|1]
Txx NAP NAN NBP NBN [**Z0**=VAL1] **F**=VAL3 [**NL**=VAL4] [**STATISTICAL**=0|1]

Lossy Transmission Line

```
Yxx LDTL [PIN:] P1...PN [REFin] PN+1...P2N REFout
+ [PARAM:] [LEVEL=val] [LENGTH=val] [SAVEFIT=val] [M=val]
```

Lossy Transmission Line: W Model

```
Wxx N=nb_line
+ P1...PN PGNDin PN+1...P2N PGNDout
+ RLGCfile=file_name L=length [FP=val]
+ [MULTIDEBYE=val] [SAVEFIT=val] [COMPAT=val] [FGD=val]
```

Lossy Transmission Line: U Model

```
Uxx P1...PN PGNDin PN+1...P2N PGNDout UNAME L=length [SAVEFIT=val]
```

MTEE—Microstrip T Junction

```
Yxx MTEE P1 P2 P3 P4 P5 P6 PARAM: [W1=val] [W2=val] [W3=val]
+ [T=val] [Er=val] [H=val]
```

MBEND—Microstrip Bend (Arbitrary Angle, Optimally Mitered)

```
Yxx MBEND P1 P2 P3 P4 PARAM: [W=val] [H=val] [Er=val] [T=val]
+ [RHO=val] [TAND=val] [M=val] [ANGLE=val]
```

MBEND2—90-degree Microstrip Bend (Mitered)

```
Yxx MBEND2 P1 P2 P3 P4 PARAM: [H=val] [W=val] [Er=val]
```

MBEND3—90-degree Microstrip Bend (Optimally Mitered)

```
Yxx MBEND3 P1 P2 P3 P4 PARAM: [W=val] [H=val] [Er=val] [T=val]
+ [RHO=val] [TAND=val]
```

MCORN—90-degree Microstrip Bend (Unmitered)

```
Yxx MCORN P1 P2 P3 P4 PARAM: [W=val] [H=val] [Er=val]
```

MSTEP—Microstrip Step in Width

```
Yxx MSTEP P1 P2 P3 P4 PARAM: [W1=val] [W2=val] [ER=val]
+ [H=val] [F=val] [ASYMMETRICAL=val] [T=val]
```

VIA2—Cylindrical Via Hole in Microstrip

```
Yxx VIA2 P1 P2 PARAM: [H=val] [R=val] [COND=val] [T=val] [F=val]
```

SBEND—Unmitered Stripline Bend

```
Yxx SBEND P1 P2 P3 P4 PARAM: [W=val] [B=val] [ER=val] [T=val]
+ [ANGLE=val] [F=val]
```

STEE—Stripline T Junction

```
Yxx STEE P1 P2 P3 P4 P4 P5 PARAM: [W1=val] [W2=val] [W3=val]
+ [B=val] [ER=val] [T=val] [F=val]
```

SSTEP—Stripline Step in Width

```
Yxx SSTEP P1 P2 P3 P4 PARAM: [W1=val] [W2=val] [B=val] [T=val]
+ [ER=val] [F=val]
```

Junction Diode

```
Dxx NP NN [NM] MNAME [[AREA=]AREA_VAL] [PERI|PJ|PD=PERIVAL]
+ [PGATE=PGATE_VAL] [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [OFF=0|1]
+ [STATISTICAL=0|1] [NOISE=0|1] [NONOISE]
Dxx NP NN [NM] MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

BJT—Bipolar Junction Transistor

```
Qxx NC NB NE [NS] [TH] MNAME [[AREA=]AREA_VAL] [AREAB=AREA_VAL]
+ [AREAC=AREA_VAL] [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [OFF=0|1]
+ [STATISTICAL=0|1] [NOISE=0|1] [NONOISE]
Qxx NC NB NE [NS] MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

JFET—Junction Field Effect Transistor

```
Jxx ND NG NS MNAME [[AREA=]AREA_VAL] [L=VAL] [W=VAL]
+ [T[EMP]=VAL] [DTEMP=VAL] [STATISTICAL=0|1] [OFF=0|1] [NONOISE]
Jxx ND NG NS MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

MESFET—Metal Semiconductor Field Effect Transistor

```
Jxx ND NG NS MNAME [AREA] [L=VAL] [W=VAL] [T[EMP]=VAL] [DTEMP=VAL]
+ [STATISTICAL=0|1] [OFF=0|1] [NONOISE]
Jxx ND NG NS MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

MOSFET

```
Mxx ND NG NS [NB] [{NN}] [MOD[EL]=]MNAME [[L=]VAL] [[W=]VAL]
+ [AD=VAL] [AS=VAL] [PD=VAL] [PS=VAL] [GEO=VAL] [NRD=VAL] [NRS=VAL]
+ [M=VAL] [RDC=VAL] [RSC=VAL] [T[EMP]=VAL] [DTEMP=VAL] [STATISTICAL=0|1]
+ [NONOISE]
Mxx ND NG NS [NB] [{NN}] [MOD[EL]=]MNAME [W=VAL] [L=VAL]
+ [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

S-Domain Filter

```
FNSxx IN OUT [RIN=val] [ROUT=val] NN {NN}, DN {DN}
```

Z-Domain Filter

```
FNZxx IN OUT FREQ=VAL [RIN=val] [ROUT=val] NN {NN}, DN {DN}
```

Subcircuit Instance

```
Xxx NN {NN} NAME [PAR=VAL] [PAR={EXPR}] [M=VAL] [TEMP=VAL]
+ [STATISTICAL=0|1] [(SWITCH|ANALOG|OSR|DIGITAL)] [NONOISE|NOISE=0]
Xxx [MODEL:] MNAME PIN: {pin=net}
+ PARAM: {par=val} KEYWORD: {keywords} [STATISTICAL=0|1]
Xxx [MODEL:] MNAME NET: {net=pin}
+ PARAM: {par=val} KEYWORD: {keywords} [STATISTICAL=0|1]
```

Sources

Independent Voltage Source

```
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TIME_DEPENDENT_FUNCTION1] [TC1=val] [TC2=val]
+ [RPORT=val [NONOISE]] [RPORT_TC1=val] [RPORT_TC2=val]
+ [IPORT=val] [CPORT=val] [LPORT=val] [MODE=keyword] [NOISETEMP=val]
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TIME_DEPENDENT_FUNCTION1] [TC1=val] [TC2=val]
+ ZPORT_FILE=string [IPORT=val] [CPORT=val] [LPORT=val] [MODE=keyword]
+ [NOISETEMP=val]
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE [THN=VAL] [FLN=VAL]
+ [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL] [FMAX=VAL]
+ [NBF=VAL]
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE [THN=VAL] [FLN=VAL]
+ [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL] [FMAX=VAL]
+ [NBF=VAL]
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG] [DB|MA]
+ (f1 val1) (f2 val2) ...
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG] [DB|MA]
+ (f1 val1) (f2 val2) ...
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] FOUR
+ fund1 [fund2 [fund3]] MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
```

1. Refer to the **EXP**, **PATTERN**, **PULSE**, **PWL**, **SFFM** and **SIN** source functions.

```
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string
+ [IPORT=val] [CPORT=val] [LPORT=val] [MODE=keyword] [NOISETEMP=val] FOUR
+ fund1 [fund2 [fund3]] MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
```

Independent Current Source

```
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TIME_DEPENDENT_FUNCTION1] [TC1=val] [TC2=val]
+ [RPORT=val [NONOISE]] [RPORT_TC1=val] [RPORT_TC2=val]
+ [IPORT=val] [CPORT=val] [LPORT=val] [NOISETEMP=val]
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TIME_DEPENDENT_FUNCTION1] [TC1=val] [TC2=val]
+ ZPORT_FILE=string [IPORT=val] [CPORT=val] [LPORT=val] [MODE=keyword]
+ [NOISETEMP=val]
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [NOISETEMP=val] NOISE [THN=VAL] [FLN=VAL]
+ [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL] [FMAX=VAL]
+ [NBF=VAL]
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [IPORT=val] [CPORT=val]
+ ZPORT_FILE=string [LPORT=val] [MODE=keyword] [NOISETEMP=val]
+ NOISE [THN=VAL] [FLN=VAL] [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL]
+ [FMAX=VAL] [NBF=VAL]
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG] (f1 val1) (f2 val2) ...
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG] (f1 val1) (f2 val2) ...
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [NOISETEMP=val] FOUR fund1 [fund2 [fund3]]
+ MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] FOUR fund1 [fund2 [fund3]]
+ MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
```

Amplitude Modulation Function

```
AM (AMPLITUDE OFFSET FM FC TD)
```

1. Refer to the **EXP**, **PULSE**, **PWL**, **SFFM** and **SIN** source functions.

Exponential Function

```
EXP (V1 V2 [TD1 [TAU1 [TD2 [TAU2]]]])
```

Noise Function

```
NOISE THN FLN ALPHA [FC N] [FMIN] [FMAX] [NBF]
```

Noise Table Function

```
NOISE TABLE [[INTERP=]DEC|OCT|LIN|LOG|HARM_DEC|HARM_OCT] [DB|MA]
+ (f1 val1) (f2 val2) ...
```

Pattern Function

```
PATTERN VHI VLO TDELAY TRISE TFALL TSAMPLE BITS R
```

Pulse Function

```
PULSE (V0 V1 [TD [TR [TF [PW [PER]]]]])
```

Piece Wise Linear Function

```
PWL (TN VN {TN VN} [TD=val] [R=val] [SHIFT=val] [R] [SCALE=val]
+ [STRETCH=val])
PWL (FILE=<pwl_file> [TD=val] [R=val] [SHIFT=val] [R] [SCALE=val]
+ [STRETCH=val])
PWL (FILE=<pwl_file> [COL=val] [ISTEP=val] [ISTART=val] [ISTOP=val]
+ [TD=val] [R=val] [SHIFT=val] [R] [SCALE=val] [STRETCH=val])
```

Single Frequency FM Function

```
SFFM (SO SA [FC [MDI [FS]]])
```

Sine Function

```
SIN (VO VA [FR [TD [THETA [PHASE]]]])
```

Trapezoidal Pulse With Bit Pattern Function

```
PBIT V0 V1 TD TD01 TR01 TD10 TF10 BITTIME {PATTERN} [R]
```

Exponential Pulse With Bit Pattern Function

```
EBIT V0 V1 TD TD01 TAU01 TD10 TAU10 BITTIME {PATTERN} [R]
```

Voltage Controlled Voltage Source

```
Exx NP NN [VCVS] NCP NCN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN [VCVS] NCP NCN VAL0 {VALn} [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

```
Exx NP NN [VCVS] POLY(ND) PCP PCN {PCP PCN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN PWL(1) NCP NCN PWL_LIST [DELTA=val]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) PCP PCN {PCP PCN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN DELAY NCP NCN [TD=val] [ABS=VAL]
Exx NP NN VALUE={EXPR} [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL]
+ TABLE EXPR=(XN YN) {(XN YN)} [ABS=VAL]
Exx NP NN INTEGRATION|DERIVATION NCP NCN VAL
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
Exx NP NN PZ  NCP NCN a zr1 zi1 ... zrm zim, b pr1 pi1 ... prn pin
Exx NP NN FREQ NCP NCN f0 a0 ph0 f1 a1 ph1... fn an phn
+ [RESTORE_CAUSALITY=val]
Exx NP NN TRANS[FORMER] NCP NCN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN OPAMP NCP NCN
```

Current Controlled Current Source

```
Fxx NP NN [CCCS] VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Fxx NP NN [CCCS] POLY(N) VN {VN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Fxx NP NN PWL(1) VN PWL_LIST [DELTA=val]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Fxx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) VN {VN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Fxx NP NN DELAY VN [TD=val] [ABS=VAL]
Fxx NP NN INTEGRATION|DERIVATION VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Voltage Controlled Current Source

```
Gxx NP NN [VCR|VCCAP|VCCS]] NCP NCN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Gxx NP NN [VCR|VCCAP|VCCS] POLY(ND) PCP PCN {PCP PCN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Gxx NP NN VCR [PWL(1)|NPWL(1)|PPWL(1)] NCP NCN PWL_LIST
+ [DELTA=val] [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL]
+ [SCALE=VAL] [ABS=VAL]
Gxx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) PCP PCN {PCP PCN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Gxx NP NN DELAY NCP NCN [TD=val] [ABS=VAL]
Gxx NP NN VALUE={EXPR} [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Gxx NP NN [VCR|VCCAP|VCCS] [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL]
+ [SCALE=VAL] [ABS=VAL] TABLE EXPR=(XN YN) {(XN YN)}
Gxx NP NN INTEGRATION|DERIVATION NCP NCN VAL
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Gxx NP NN FREQ NCP NCN f0 a0 ph0 f1 a1 ph1... fn an phn
+ [RESTORE_CAUSALITY=val]
```

Current Controlled Voltage Source

```
Hxx NP NN [CCVS] VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Hxx NP NN [CCVS] POLY(N) VN {VN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Hxx NP NN PWL(1) VN PWL_LIST [DELTA=val]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Hxx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) VN {VN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Hxx NP NN DELAY VN [TD=val] [ABS=VAL]
Hxx NP NN INTEGRATION|DERIVATION VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

S, Y, Z Parameter Extraction

```
Vyy NP NN IPORT=VAL [RPORT=VAL] [CPORT=VAL] [LPORT=VAL] [MODE=KEYWORD]
Vyy NP NN IPORT=VAL ZPORT_FILE=string [CPORT=VAL] [LPORT=VAL]
+ [MODE=KEYWORD]
Iyy NP NN IPORT=VAL [RPORT=VAL] [CPORT=VAL] [LPORT=VAL] [MODE=KEYWORD]
Iyy NP NN IPORT=VAL ZPORT_FILE=string [CPORT=VAL] [LPORT=VAL]
+ [MODE=KEYWORD]
```

Macromodels

Analog

Comparator

```
COMPxx INP INN OUT [MNAME] [VHI=VAL1] [VLO=VAL2]
+ [VOFF=VAL3] [VDEF=VAL4] [TCOM=VAL5] [TPD=VAL6]
COMPDxx INP INN OUTP OUTN [MNAME] [VHI=VAL1] [VLO=VAL2]
+ [VOFF=VAL3] [VDEF=VAL4] [TCOM=VAL5] [TPD=VAL6]
```

Op-amp (Linear)

```
Yxx OPAMP0 [PIN:] INP INN OUT AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx OPAMP0D [PIN:] INP INN OUTN OUTP AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Op-amp (Linear 1-pole)

```
Yxx OPAMP1 [PIN:] INP INN OUT AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx OPAMP1D [PIN:] INP INN OUTN OUTP AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Op-amp (Linear 2-pole)

```
Yxx OPAMP2 [PIN:] INP INN OUT AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

```
Yxx OPAMP2D [PIN:] INP INN OUTN OUTP AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Delay

```
DELxx IN OUT VAL
```

Saturating Resistor

```
Yxx SATR [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Voltage Limiter

```
Yxx SATV [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Current Limiter

```
Yxx SATI [PIN:] IN OUT
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Voltage Controlled Switch

```
Yxx VSWITCH [PIN:] NP NN CP CN [PARAM: PAR=VAL {PAR=VAL}] MODEL: MNAME
```

Current Controlled Switch

```
Yxx CSWITCH [PIN:] NP NN IC: VNAME
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Triangular to Sine Wave Converter

```
Yxx TRI2SIN [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Staircase Waveform Generator

```
Yxx STAIRGEN [PIN:] NP NN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Sawtooth Waveform Generator

```
Yxx SAWGEN [PIN:] NP NN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Triangular Waveform Generator

```
Yxx TRIGEN [PIN:] NP NN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Amplitude Modulator

```
Yxx AMM [PIN:] INP INN OUTP OUTN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Pulse Amplitude Modulator

```
Yxx PAM [PIN:] INP INN OUTP OUTN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Sample and Hold

```
Yxx SA_HO [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Track and Hold

```
Yxx TR_HO [PIN:] INP INN OUTP OUTN CRT
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Pulse Width Modulator

```
Yxx PWM [PIN:] CTRP CTRN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Voltage Controlled Oscillator

```
Yxx VCO [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Peak Detector

```
Yxx PEAK_D [PIN:] INP INN OUTP OUTN CRT
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Level Detector

```
Yxx LEV_D [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx LEV_D [PIN:] INP INN OUTP OUTN REF
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Logarithmic Amplifier

```
Yxx LOGAMP [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Anti-logarithmic Amplifier

```
Yxx EXPAMP [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Differentiator

```
Yxx DIFF [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Integrator

```
Yxx INTEG [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Adder, Subtractor, Multiplier and Divider

```
Yxx ADD [PIN:] IN1 IN2 OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx SUB [PIN:] IN1 IN2 OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx MULT [PIN:] IN1 IN2 OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx DIV [PIN:] IN1 IN2 OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Digital

Digital Model Definition

```
.MODEL MNAME LOGIC [VHI=VAL1] [VLO=VAL2] [VTH=VAL3]
+ [VTHI=VAL4] [VTLO=VAL5] [TPD=VAL6] [TPDUP=VAL7]
+ [TPDOWN=VAL8] [CIN=VAL9] [DRVL=VAL10] [DRVH=VAL11]
```

Delay

```
DELxx IN OUT VAL
```

Inverter

```
INVxx IN OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Exclusive-OR Gate

```
XORxx IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

2-Input Digital Gates

```
<dgate><xx> IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Nand   NANDxx   IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
And    ANDxx    IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Nor    NORxx    IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Or     ORxx     IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Xor    XORxx    IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

3-Input Digital Gates

```
<dgate><xx> IN1 IN2 IN3 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Nand   NAND3xx     IN1 IN2 IN3 OUT [REF1 REF2] [MNAME] [PAR=VAL]
And    AND3xx      IN1 IN2 IN3 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Nor    NOR3xx      IN1 IN2 IN3 OUT [REF1 REF2] [MNAME] [PAR=VAL]
Or     OR3xx       IN1 IN2 IN3 OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Multiple Input Digital Gates

```
<dgate><xx> IN1 IN2... {INX} OUT [REF1 REF2] [MNAME] [PAR=VAL]
Nand   NAND#xx     IN1 IN2..{INX} OUT [REF1 REF2] [MNAME] [PAR=VAL]
And    AND#xx      IN1 IN2..{INX} OUT [REF1 REF2] [MNAME] [PAR=VAL]
Nor    NOR#xx      IN1 IN2..{INX} OUT [REF1 REF2] [MNAME] [PAR=VAL]
Or     OR#xx       IN1 IN2..{INX} OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Mixed

Analog to Digital Converter

```
ADCxx CLK IN OUTSB{OUTSB} [EDGE=VAL1] [VTH=VAL2] [VHI=VAL3]
+ [VLO=VAL4] [VINF=VAL5] [VSUP=VAL6] [TCOM=VAL7] [TPD=VAL8]
```

Digital to Analog Converter

```
DACxx CLK INSB{INSB} OUT [EDGE=VAL1] [VTH=VAL2]
+ [VTIN=VAL3] [VHI=VAL4] [VLO=VAL5] [TPD=VAL6] [SL=VAL7]
```

Magnetic

Transformer Winding

```
Yxx WINDING [PIN:] E1 E2 M1 M2 [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Non-linear Magnetic Core 1

```
Yxx NLCORE1 [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Non-linear Magnetic Core 2

```
Yxx NLCORE2 [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Linear Magnetic Core

```
Yxx LINCORE [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Magnetic Air Gap

```
Yxx AIRGAP [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Transformer (Variable # of Windings)

```
Yxx LVTRANS [PIN:] P1P P1N P2P P2N {PNP PNN}
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Ideal Transformer

```
Yxx JTRAN N1 N2 N3 N4 [PARAM: A=VAL]
```

Switched Capacitor

Operational Amplifier

```
OPAxx INP INN OUTP OUTN [MNAME] [LEVEL=VAL1] [VOFF=VAL2]
+ [SL=VAL3] [CIN=VAL4] [RS=VAL5] [VSAT=VAL6] [VSATN=VAL7] [GAIN=VAL8]
+ [FC=VAL9] [FNDP=VAL10] [IMAX=VAL11] [CMRR=VAL12]
```

Switch

```
Sxx NC N1 N2 [MNAME] [RON [CREC]]
```

Ideal Operational Amplifier

```
Yxx SC_IDEAL [PIN:] INP INN OUT [PARAM: M=val]
```

Inverting Switched Capacitor

```
Yxx SC_I [PIN:] P1 P2 N2 N1 [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Non-inverting Switched Capacitor

```
Yxx SC_N [PIN:] P1 P2 N1 N2 [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Parallel Switched Capacitor

```
Yxx SC_P [PIN:] P1 P2 N [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Serial Switched Capacitor

```
Yxx SC_S1 [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx SC_S2 [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Serial-parallel Switched Capacitor

```
Yxx SC_SP1 [PIN:] IN OUT REF [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx SC_SP2 [PIN:] IN OUT REF [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Bi-linear Switched Capacitor

```
Yxx SC_B [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Unswitched Capacitor

```
Yxx SC_U [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Commands

Analog-to-Digital Converter

```
.A2D [SIM=simulator] eldo_node_name
+ [digital_node_name] [MOD=model_name] [parameters_list]
```

AC Analysis

```
.AC TYPE nb fstart fstop [SWEEP DATA=dataname] [UIC] [MONTE=val]
.AC TYPE nb fstart fstop [SWEEP parameter_name TYPE nb start stop]
+ [UIC] [MONTE=val]
.AC TYPE nb fstart fstop [SWEEP parameter_name start stop incr]
```

```
+ [UIC] [MONTE=val]
.AC DATA=dataname [SWEEP DATA=dataname] [UIC] [MONTE=val]
.AC DATA=dataname [SWEEP parameter_name TYPE nb start stop]
+ [UIC] [MONTE=val]
.AC DATA=dataname [SWEEP parameter_name start stop incr] [UIC] [MONTE=val]
.AC LIST {list_of_frequency_points}
+ [SWEEP DATA=dataname] [UIC] [MONTE=val]
.AC LIST {list_of_frequency_points}
+ [SWEEP parameter_name TYPE nb start stop] [UIC] [MONTE=val]
.AC LIST {list_of_frequency_points}
+ [SWEEP parameter_name start stop incr] [UIC] [MONTE=val]
.AC ADAPTIVE tolerance_value fstart fstop
```

Insert a Model or Subcircuit File

```
.ADDLIB N DIR_NAME
```

Age Analysis

```
.AGE [TAGE=value] [TUNIT=year|month|day|hour|min|sec] [NBRUN[S]=value]
+ [LIN[={YES|ON|1}|{NO|OFF|0}] | LOG[={YES|ON|1}|{NO|OFF|0}]]
+ [LOGMODE_MINEXP=value]
+ [TSTART=value] [TSTOP=value] [TWINDOW={(a1,b1) (an,bn)}]
+ [MODE= sim | load | save | MCload | blockload]  [AGELIB=file_name]
+ [AGEALL[={YES|ON|1}|{NO|OFF|0}]]
+ [RESTRICT_MC={YES|ON|1}|{NO|OFF|0}]
+ [ASCII[={YES|ON|1}|{NO|OFF|0}]]
+ [COMPUTE_LAST[={YES|ON|1}|{NO|OFF|0}]]
+ [PLOT={FRESH_FINAL|ALL}]
+ [AGEDSIM={YES|ON|1}|{NO|OFF|0}]
+ [PRINT_CONFIGURATION={YES|ON|1}|{NO|OFF|0}]
+ [STRESS_LIST={YES|ON|1}|{NO|OFF|0}]
+ [STRESS_SORT_NBMAX=value]
+ [STRESS_SORT_REL=value]
+ [STRESS_SORT_ABS=value]
+ [STRESS_LIST_FILE=file_name]
+ [STRESS_LIST_SPLIT_MOS={YES|ON|1}|{NO|OFF|0}]
+ [STRESS_SORT_ORDER=ASCENDING|DESCENDING]
+ [STRESS_SORT_DELTA=extract_label]
+ [DELTA_VDS=value]
+ [DELTA_VGS=value]
+ [DELTA_VBS=value]
+ [USER_WARNING={YES|ON|1}|{NO|OFF|0}]
```

Define Functions For Reliability

```
.AGE_LIB
+ FILE=library_name
+ FNC_PREFIX=fnc_prefix
+ LEVEL=level1[{, level2, ..., leveln}]
+ [SHARED_PATH=yes|no]
+ [PFDIR=library_path]
```

Reliability Model Parameter Declaration

```
.AGEMODEL MODEL=model_name [parameter=value]
```

Generalized Re-run Facility

```
.ALTER [LABEL]
[ELEMENT]
[SUBCKT]
[COMMAND]
[COMMENT]
.ALTER | .END
```

Configure Spice Descriptions

```
.BIND INST=inst_name [EXCEPT=inst_name]
| FROM_SUBCKT=Eldo_subckt_name | FROM_MODEL=HDL_model_name
  TO_SUBCKT=new_Eldo_subckt_name
  [FILE=<file_name> VARIANT=<variant_name>]
| TO_MODEL=new_HDL_model_name
 [MAPPING=assoc_file_name | DEFAULT_MAPPING=by_name|by_position]
.BIND INST=inst_name [EXCEPT=inst_name]
  FROM_SUBCKT=Eldo_subckt_name | FROM_MODEL=HDL_model_name
  TO_SUBCKT=new_Eldo_subckt_name
  [FILE=<file_name> VARIANT=<variant_name>]
| TO_MODEL=new_HDL_model_name
 [MAPPING=assoc_file_name | DEFAULT_MAPPING=by_name|by_position]
```

Define local scope for .BIND

```
.BINDSCOPE PATH=scope_path
```

Call Tcl Function

```
.CALL_TCL [TRAN|AC|DC|...]
+ WHERE=START|START_OF_RUN|END_OF_RUN|END
+ [PLOT=[YES|NO|0|1]] [PLOT_TYPE=[TRAN|AC|DC|...]]
+ [LABEL=alias_name] tcl_function_call
```

Check Bus Values

```
.CHECKBUS BNAME [VTH[1]=VAL1] [VTH2=VAL2] [BASE=DEC|OCT|BIN|HEX] [LOCK=1]
+ [REPORTX=0|1] TN VAL {TN VAL}
.CHECKBUS BNAME [VTH[1]=VAL1 [VTH2=VAL2]] [TSAMPLE=VAL] [TDELAY=VAL]
+ [BASE=DEC|OCT|BIN|HEX] [LOCK=1] [REPORTX=0|1] PATTERN BITS {BITS}
```

Check Safe Operating Area Limits

```
.CHECKSOA [ANALYSIS] [TSTART=val1 [TSTOP=val2]] [AUTOSTOP]
+ [NOMERGE] [NOLIB] [FILE=file_name] [NOXWINDOW]
+ [SUBCKT={list_of_subckt_instances}] [RUNTMSG]
```

Piece Wise Linear Source

```
.CHRENT NODE TN VN {TN VN} [P|F]
.CHRENT NODE (TN VN {TN VN}) FACTN {(TN VN {TN VN}) FACTN} [P|F]
```

Input from a Prior Simulation

```
.CHRSIM IN OUT FILE [TSTART=V1] [TSTEP=V2] [BP=0|1|2]
+ [ZOOMTIME] [FORMAT=WDB] [RUN=val] [TYPE=CURRENT|VOLTAGE]
```

Change Comment Character

```
.COMCHAR char {char}
```

Connect Two Nodes

```
.CONNECT N1 N2
```

Current Used by a Circuit

```
.CONSO VN {VN}
```

Correlation between Parameters

```
.CORREL [PARAM=]param_list cc=VAL
.CORREL DEV[ICE]=device_list PARAM=param_list cc=VAL
```

Digital-to-Analog Converter

```
.D2A [SIM=simulator] eldo_node_name
+ [digital_node_name] [MOD=model_name] [parameters_list]
```

Parameter Sweep

```
.DATA dataname parameter_list
[+] val_list1
[+] val_list2
[+] ...
[.ENDDATA]
.DATA dataname MER[GE]|LAM[INATED]
[+] file=filename1 param=column ... param=column
[+] file=filename2 param=column ...
[+] ...
[+] [out=outfile]
[.ENDDATA]
```

DC Analysis

```
.DC
.DC CNAM [L|W] [TYPE nb] START STOP INCR [SWEEP DATA=dataname] [MONTE=val]
.DC CNAM [L|W] [TYPE nb] START STOP INCR
+ [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC CNAM [L|W] [TYPE nb] START STOP INCR
+ [SWEEP parameter_name start stop incr] [MONTE=val]
.DC SNAM [TYPE nb] START STOP INCR [SNAM2 START2 STOP2 INCR2]
+ [SWEEP DATA=dataname] [MONTE=val]
.DC SNAM [TYPE nb] START STOP INCR [SNAM2 START2 STOP2 INCR2]
+ [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC SNAM [TYPE nb] START STOP INCR [SNAM2 START2 STOP2 INCR2]
```

```
+ [SWEEP parameter_name start stop] incr [MONTE=val]
.DC TEMP START STOP INCR [SWEEP DATA=dataname] [MONTE=val]
.DC TEMP START STOP INCR [SWEEP parameter_name TYPE nb start stop]
+ [MONTE=val]
.DC TEMP START STOP INCR [SWEEP parameter_name start stop incr]
+ [MONTE=val]
.DC PARAM PARAM_NAME START STOP INCR [SWEEP DATA=dataname]
+ [MONTE=val]
.DC PARAM PARAM_NAME START STOP INCR
+ [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC PARAM PARAM_NAME START STOP INCR
+ [SWEEP parameter_name start stop incr] [MONTE=val]
.DC PARAM PARAM_NAME [TYPE nb] START STOP INCR
.DC DATA=dataname [SWEEP DATA=dataname] [MONTE=val]
.DC DATA=dataname [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC DATA=dataname [SWEEP parameter_name start stop incr] [MONTE=val]
```

DC Mismatch Analysis

```
.DCMISMATCH [output] [DCALL[=0|1]]
+ [SORT_REL=value] [SORT_ABS=value] [SORT_NBMAX=value] [NSIGMA=value]
```

Set Default Conditions

```
.DE[FAULT] TYPE VALUE
.DE[FAULT] TYPE {KEYWORD [VALUE]}
```

Macro Definition

```
.DEFMAC MAC_NAME(ARG{, ARG})=EXPRESSION
```

Model Name Mapping

```
.DEFMOD alias_model_name actual_model_name
```

Plotting an Analog Signal as a Digital Bus

```
.DEFPLOTDIG [VTH[1]=VAL1 [VTH2=VAL2]]
```

Waveform Definition

```
.DEFWAVE [SWEEP] [ANALYSIS] WAVE_NAME=WAVE_EXPR
```

Remove library name

```
.DEL LIB LIB_NAME
```

Design of Experiments

```
.DEX
+ EXPERIMENT = SCREENING | SCREENING_CTRL | SCREENING_NOISE
+ RESPONSE = LIST_OF_MEASURES
+ [DESIGN = ORTHA_2_N | ORTHA_2_2N | FULL_FACT]
+ [FACTOR = LIST_OF_FACTORS]
```

```
+ [FIND_FACTOR]
```

Ignore Instances or Subckt Definitions

```
.DISCARD INST | SUBCKT | DEV =(NAME, {NAME})
```

Disable Flat Netlist Mode

```
.DISFLAT
```

User Defined Distributions (Monte Carlo)

```
.DISTRIB DIST_NAME (DEV1 PROB1) [{(DEVn PROBn)}]
```

DSP (Digital Signal Processing) Computation

```
.DSP LABEL=label_name MODEL=model_name waveform_name
```

Load DSPF File

```
.DSPF_INCLUDE [FILE=]DSPF_FILENAME [INST={list_of_subckt_inst}]
+ [LEVEL=C|RC|RCC] [DEV=DSPF|SCH[EMATIC]] [RMINVAL=val] [CMINVAL=val]
+ [CCMINVAL=val] [ADDXNET] [ADDX] [MSUFFIX=string]
.DSPF_INCLUDE [FILE=]DSPF_FILENAME [LEVEL=C|RC|RCC]
+ [DEV=DSPF|SCH[EMATIC]] DEDICATEDX=subckt_name [RMINVAL=val]
+ [CMINVAL=val] [CCMINVAL=val] [ADDXNET] [ADDX] [MSUFFIX=string]
```

PSD (Power Spectral Density) Computation

```
.DSPMOD DSP=CORRELO|PERIODO LABEL=label_name
+ [TSTART=val] [TSTOP=val] [FS=val] [NBPT=val]
+ [PADDING=val] [WINDOW=name] [ALPHA=val] [BETA=val]
+ [NORMALIZED=val] [INTERPOLATE=val] [DISPLAY_INPUT=val]
+ [FNORMAL=val] [FMIN=val] [FMAX=val]
+ [NAUTO=val] [NCORR=val] [NPSD=val] [NSECT=val]
```

Histogram Computation

```
.DSPMOD DSP=HISTOGRAM LABEL=label_name NBINTERVAL=val
+ [XSTART=val XSTOP=val SAMPLE=YES|NO FS=val]
```

Frequency Computation

```
.DSPMOD DSP=FREQUENCY LABEL=label_name
+ [BASELINE=val] [TOPLINE=val]
+ [EDGE_TRIGGER = RISING | FALLING | EITHER]
+ [XSTART=val] [XEND=val]
```

End Eldo Netlist

```
.END
```

End Eldo Library Variant Description

```
.ENDL
```

End Eldo Subcircuit Description

```
.ENDS
```

Replace Node Name for Display

```
.EQUIV new_name=netlist_name
```

Extract Mode

```
.EXTMOD FILE=filename
```

Extract Waveform Characteristics

```
.EXTRACT [EXTRACT_INFO] [LABEL=NAME] [FILE=FNAME] [VECT]
+ [CATVECT] $MACRO|FUNCTION [OPTIMIZER_INFO] [MC_INFO] [TIME=VALUE]
+ [INTERP_MODE=LINEAR|QUADRATIC|SAMPHOLD|HISTOGRAM|SPECTRAL]
```

S, Y, Z Parameter Output File Specification

```
.FFILE S|Y|Z|G|H|T|A [SINGLELINE] FILENAME [HZ|KHZ|MHZ|GHZ] [RI|MA|DB]
```

Filter Data Driven Simulations

```
.FILTER AC|DC|TRAN|MODSST|SST|ALL [condition]
.FILTER EXTRACT [LABEL=label] [condition]
.FILTER DATA DATA_IN=input_data DATA_OUT=output_data [condition]
```

Initial Transient Analysis Conditions

```
.FORCE [NODE] {node_name value}
+ [IND|OBJ] {object_name value}
```

FFT Select Waveform

```
.FOUR LABEL = label_name waveform_name
```

User Defined Function

```
.FUNC P(a,b,...) EXPR
```

Global Node Allocation

```
.GLOBAL NN {NN}
```

Initial DC Analysis Conditions

```
.GUESS V(NN)=VAL [SUBCKT=subckt_name] {V(NN)=VAL [SUBCKT=subckt_name]}
```

Changing the Hierarchy Separator

```
.HIER .|/|<char>
```

Initial Transient Analysis Conditions

```
.IC V(NN)=VAL [SUBCKT=subckt_name] {V(NN)=VAL [SUBCKT=subckt_name]}
```

Ignore DSPF on Specified Node

```
.IGNORE_DSPF_ON_NODE {NODE}
```

Include a File in an Input Netlist

```
.INC[LUDE] FNAME
```

Initial Digital Circuit Conditions

```
.INIT NODE [DC=VAL] TI TS VALI {TI TS VALI}
```

Current Probe

```
.IPROBE LABEL=vname [DIRECTION=POS|NEG ] pinref1 {pinrefn}
```

Insert Circuit Information from a Library File

```
.LIB [KEY=KNAME] FNAME [LIBTYPE]
.LIB LIBTYPE
```

Use Previously Simulated Results

```
.LOAD [FILE=]filename
```

Insert a Feedback Loop

```
.LOOP INPUT OUTPUT [R|C|I|V VALUE] [DISCONNECT=DEV_NAME] [KEEPINPUT]
```

Share Distributions

```
.LOTGROUP group_name[/distrib_type]=val[%]
```

Loop Stability Analysis

```
.LSTB SOURCE_NAME
```

Model and Subcircuit Name Mapping

```
.MALIAS alias_model_name actual_model_name
```

Map Eldo Node to DSPF Node

```
.MAP_DSPF_NODE_NAME LOGICAL=ELDONAME DSPF=NEWNAME
```

Monte Carlo Analysis

```
.MC RUNNO [OUTER] [OV] [SEED=integer_value] [NONOM] [ALL]
+ [VARY=LOT|DEV] [IRUN=val] [NBBINS=val] [ORDMCS] [MCLIMIT]
+ [PRINT_EXTRACT=NOMINAL|ALL|run_number] [SIGBIN=val]
+ [MAXABSBIN=val] [MAXRELBIN=val]
```

LOT & DEV Variation Specification on Model Parameters (Monte Carlo)

```
.MCMOD MNAME [(list_of_instances)] PAR LOT|DEV=VAL {PAR LOT|DEV=VAL}
.MCMOD MNAME PAR LOTGROUP=my_lot_group
```

Measure Waveform Characteristics

```
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name
+ TRIG trig_spec TARG targ_spec
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name WHEN when_spec AT val
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name FIND wave WHEN when_spec
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name FIND wave AT val
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name FIND W(‘wave’)
+ WHEN when_spec [FROM=val] [TO=val]
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name DERIVATIVE wave
+ WHEN when_spec
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name DERIVATIVE wave AT val
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name meas_k wave
+ [FROM=val] [TO=val]
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name PARAM='expression'
```

Aspire/SimPilot Command

```
.MODDUP device_name [... device_name]
.MODDUP element_name
```

Device Model Description

```
.MODEL MNAME TYPE NONOISE [PAR=VAL]
.MODEL LIB FILENAME MODNAME [LIBTYPE]
```

Monitor Simulation Steps

```
.MONITOR ANALYSIS [=] [modulo]
```

Multi-Processor Simulation

```
.MPRUN [ALL|HOST={host[(nbjobs)]}|FILE=filename}] [NBLICENSES=val]
+ [MAX_NBJOBS=val] [CLEAN=YES|NO] [QUEUE=YES|NO] [SETENV=YES|NO]
+ [VIEW_COMMAND=YES|NO] [CHECK_DELAY=val] [INIT_FILE=filename]
+ [DEFAULT_INIT=YES|NO] [NETWORK_DIR=directory]
+ [CD_WORKDIR=YES|NO] [LOGFILE=YES|NO]
+ [SHELL_SYNTAX=(source_cmd, setenv_cmd, setenv_sep, init_anacad_ext)]
+ [USE_LOCAL_HOST=YES|NO] [FLAT=YES|NO]
+ [FILE_PREFIX=(name1,name2,...,nameX)]
+ [USE_SSH=YES|NO] [SSH_OPTIONS="{<options>}"]
+ [CHECK_ALL_HOSTS=YES|NO]
```

```
+ [SYNCHRO_NOMINAL_MC=YES|NO]
.MPRUN DISPATCHER= [LSF |
+ (dispatcher_name, install_check_cmd, submission_cmd]
+ [REMOVE_QUOTE=YES|NO] [DISPATCHER_OPTIONS=options]
+ [NBLICENSES=val] [MAX_NBJOBS=val] [CLEAN=YES|NO]
+ [QUEUE] [SETENV] [VIEW_COMMAND] [CHECK_DELAY=val]
+ [INIT_FILE=filename [DEFAULT_INIT=YES|NO]]
+ [USE_LOCAL_HOST=YES|NO] [FLAT=YES|NO]
+ [FILE_PREFIX=(name1,name2,...,nameX)]
+ [LSF_JOB_PREFIX=lsf_prefix]
.MPRUN
+ DISPATCHER_TEMPLATE=command_line
+ [USE_LOCAL_HOST=YES|NO] [FLAT=YES|NO]
+ [FILE_PREFIX=(name1,name2,...,nameX)]
```

Automatic Model Selection

```
.MSEL[LECT] dummy [MODELS] mod1 [mod2 [mod3 [...]]]
```

Network Analysis

- 2-port network

  ```
  .NET output input RIN=val ROUT=val
  ```

- 1-port network

  ```
  .NET input RIN=val
  ```

Control Page Layout

```
.NEWPAGE
```

Suppress Comment Lines from Output File

```
.NOCOM
```

DC Analysis Conditions

```
.NODESET V(NN)=VAL [SUBCKT=subckt_name] {V(NN)=VAL [SUBCKT=subckt_name]}
```

Noise Analysis

```
.NOISE OUTV INSRC NUMS
```

Transient Noise Analysis

```
.NOISETRAN FMIN=VAL FMAX=VAL NBRUN=VAL [NBF=VAL] [AMP=VAL]
+ [SEED=VAL] [NOMOD=VAL] [NONOM] [TSTART=VAL] [TSTOP=VAL]
+ [MRUN] [ALL] [NBBINS=VAL] [FMIN_FLICKER=VAL]
```

Suppress Netlist from an Output File

```
.NOTRC
```

Partition Netlist into Newton Blocks

```
.NWBLOCK [RELTOL=value] [VNTOL=value] list_of_nodes
```

Optimization Objectives

```
.OBJECTIVE
+ EXTRACT_INFO [LABEL=NAME]
+ {$MACRO|FUNCTION}
+ OBJECTIVE_INFO
+ [SCALING_INFO]
+ [PRINT_INFO]
```

DC Operating Point Calculation

```
.OP [[KEYWORD] T1 {[KEYWORD] TN}]
.OP TIME=VAL|END [STEP=VAL] [TEMP=VAL]
.OP DC=VAL [DC2=VAL] [STEP=VAL] [TEMP=VAL]
```

DC Operating Point Display

```
.OP_DISPLAY
+ [MODEL=model_name]
+ [VTMOD=0|1|2]
+ [VTVDS=vds]
+ [VTCIDS=ids_norm]
+ [VTVGSMIN=vgsmin]
+ [VTVGSMAX=vgsmax]
```

FFT Post-processor Options

```
.OPTFOUR [TSTART=VAL|EXPR] [TSTOP=VAL|EXPR] [NBPT=VAL] [FS=VAL]
+ [NORMALIZED=0|1] [INTERPOLATE=0|1|2|3] [NOROUNDING[=1]] [RAPWIN=VAL]
+ [WINDOW=name] [ALPHA=VAL] [BETA=VAL] [PADDING=1|2|3]
+ [FMIN=VAL] [FMAX=VAL] [FNORMAL=freq] [DISPLAY_INPUT=0|1]
```

Optimization

```
.OPTIMIZE [qualifier=value {, qualifier=value }]
+ [PARAM=list_of_parameters | *] [RESULTS=list_of_targets | *]
```

Simulator Configuration

```
.OPT[ION] OPTION[=VAL] {OPTION[=VAL]}
```

AC Noise Analysis

```
.OPTNOISE [ALL ON|OFF] [<CLASS> ON|OFF]
+ [R ON|OFF|<max>] [OUTSOURCE ON|OFF] [NSWEIGHT <FILENAME>]
+ [SORT D|V|TD|TV [SN <n>|SV <value>]] [NBW <FMIN> <FMAX>]
```

Accuracy by Time Window

```
.OPTPWL PARAM=((TIME1,VALUE1) (TIME2,VALUE2)...)
+ PARAM=((TIME1,VALUE1)...))
```

Accuracy by Time Window

```
.OPTWIND PARAM=(TIME1,TIME2,VALUE1)... (TIME1N,TIME2N,VALUEN)
```

Global Declarations

```
.PARAM PAR=VAL {PAR=VAL}
.PARAM PAR=EXPR {PAR=EXPR}
.PARAM PAR="NAME"
.PARAM PAR(a,b)=EXPR
.PARAM PAR=VAL|PAR=EXPR
+ LOT|DEV[/GAUSS|/UNIFORM|/USERDIST]=VAL|(dtype,-3sig,+3sig
+ [,bi,-dz,+dz [,off,sv] [,scale])
.PARAM PAR=VAL LOTGROUP=my_lot_group
.PARAM PAR=MC_DISTRIBUTION
.PARAM PAR=VAL DEVX=VAL
```

Statistical Parameter Declarations

```
.PARAMDEX FACTOR_NAME [NOISE/] SIG=NSIGMA
.PARAMDEX FACTOR_NAME [NOISE/] RNG=NRANGE
.PARAMDEX FACTOR_NAME [NOISE/] ABS=DELTA
.PARAMDEX FACTOR_NAME [NOISE/] REL=DELTA%
.PARAMDEX FACTOR_NAME [CTRL/] ABS=DELTA [,NOM=RVALUE]
.PARAMDEX FACTOR_NAME [CTRL/] REL=DELTA% [,NOM=RVALUE]
```

Optimization Parameter Declarations

```
.PARAMOPT VARIABLE_NAME=(
+ [INIT_VALUE,]
+ {LOWER_BOUND | LOWER_PERCENT% },
+ {UPPER_BOUND | UPPER_PERCENT% }
+ [, INCREMENT])
```

Circuit Partitioning

```
.PART ADIT|ELDO|MODSST
+ INST=(<instance_list>) SUBCKT=(<subcircuit_list>)
```

Plotting of Simulation Results

```
.PLOT [ANALYSIS] OVN [(LOW, HIGH)] [(VERSUS)]
+ {OVN [(LOW, HIGH)]} [UNIT=NAME] [(SCATTERED)] [STEP=value]
.PLOT AC|FSST S(i, j) [(SMITH[,zref])] [(POLAR)]
.PLOT FOUR FOURxx(label_name) [(SPECTRAL)] [(CONTINUOUS)]
.PLOT DSP DSPxx(label_name)
.PLOT EXTRACT [MEAS | SWEEP]
.PLOT [CONTOUR] MEAS(meas_name_x) MEAS(meas_name_y) [(SCATTERED)]
+ [(SMITH[,zref])] [(POLAR)]
.PLOT [ANALYSIS] TWO_PORT_PARAM [(SMITH[,zref])] [(POLAR)]
```

Plotting of Bus Signals

```
.PLOTBUS BNAME [VTH[1]=VAL1 [VTH2=VAL2]]
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP] [(ANALOG)]
.PLOTBUS BNAME[MSB:LSB]|BNAME<MSB:LSB>|BNAMEMSB:LSB
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP] [(ANALOG)]
```

Printing of Results

```
.PRINT [ANALYSIS] [alias_name=]OVN
```

Printing of Bus Signals

```
.PRINTBUS BNAME [VTH[1]=VAL1 [VTH2=VAL2]]
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP]
.PRINTBUS BNAME[MSB:LSB]|BNAME<MSB:LSB>|BNAMEMSB:LSB
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP]
```

Print Tabular Output File

```
.PRINTFILE [ANALYSIS] OVN FILE=filename [STEP=value]
+ [START=value] [STOP=value] [FORMAT=DATA]
```

Output Shortform

```
.PROBE [ANALYSIS] [ALL|I|IX|ISUB|PORT|PRINT|SG|SPARAM|S|Q|V|VN|VTOP|
+ VX|VXN|W|WTOP] [MASK=mask_name] [EXCEPT=pattern] [PRINT] [STEP=val]
+ [DELTA=val]
.PROBE [ANALYSIS] [MASK=mask_name] [EXCEPT=pattern] [alias_name=] OVN
+ [PRINT] [STEP=val] [DELTA=val]
```

Printing of Bus Signals

```
.PROBEBUS BNAME [VTH[1]=VAL [VTH2=VAL]]
.PROBEBUS BNAME[MSB:LSB]|BNAME<MSB:LSB>|BNAMEMSB:LSB
```

Netlist Protection

```
.PROTECT
```

Pole-Zero Analysis

```
.PZ OV
```

Automatic Ramping

```
.RAMP DC VAL [SIMPLIFY]
.RAMP TRAN T1 T2 [SIMPLIFY]
```

Restart Simulation

```
.RESTART FNAME [FILE=WNAME]
.RESTART ["fileBasename"] [NEWEST|LONGEST|TIME=VALUE] [FILE=WNAME]
```

Save Simulation Run

```
.SAVE [[FILE=]FNAME] DC|END|TIME=VAL1 [REPEAT] [ALT|SEQ]
+ [TEMP=VAL2] [STEP=VAL3] [TYPE=NODESET|IC] [LEVEL=ALL|TOP] [CARLO=index]
```

Automatic Scaling of Active Devices

```
.SC[ALE] ELTYPE KEYWORD VALUE [KEYWORD VALUE ...]
+ [ELEMENTS ALL | EXCEPT] [ELNAME1 ELNAME2 ...] [(ELNAME1 ELNAME2)]]
.SC[ALE] ELTYPE KEYWORD VALUE [KEYWORD VALUE ...]
+ MODELS MODNAME1 [MODNAME2 ...]
.SC[ALE] MODTYPE KEYWORD VALUE [KEYWORD VALUE ....]
+ [MODELS ALL |EXCEPT] [MODNAME1 MODNAME2] [(MODNAME1 MODNAME2...)]]
.SC[ALE] P FACTOR=VALUE [SUBCKT=SUBNAME]
+ [PARAMS ALL | EXCEPT] [PARAM1 PARAM2 ...] [(PARAM11 PARAM22)]]
```

Select DSPF on Specified Node

```
.SELECT_DSPF_ON_NODE {NODE}
```

DC Sensitivity Analysis

```
.SENS OVN {OVN}
```

AC Sensitivity Analysis

```
.SENSAC OVN {OVN} FREQ=val1[{,val2}] [SORT_REL] [SORT_ABS] [SORT_MAX]
```

Sensitivity Analysis

```
.SENSPARAM sub[ckt]=subckt_name param=parameter_list
+ [var[iation]=value] [inst[ance]=instance_list]
+ [sort=inc[reasing] | dec[reasing] | alpha[betical]]
+ [sort_nbmax=value] [sort_abs=value | sort_rel=value]
```

Create Bus

```
.SETBUS BNAME PN {PN}
```

Set Reliability Model Key (Password)

```
.SETKEY [MODEL=model_name] KEY=key_value
```

Set Safe Operating Area

```
.SETSOA [LABEL="<STRING>"] [ANALYSIS]
+ E {EXPRESSION=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="<STRING>"] [ANALYSIS]
+ E IF(EXPR) THEN({PARAM=(MIN,MAX[,XAXIS])})
+ ELSE({PARAM=(MIN,MAX[,XAXIS])}) ENDIF
.SETSOA [LABEL="<STRING>"] [ANALYSIS]
+ D DNAME [SUBCKT=subckt_list|INST=inst_list] {PARAM=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="<STRING>"] [ANALYSIS]
+ D DNAME [SUBCKT=subckt_list|INST=inst_list]
```

```
+ IF(EXPR) THEN({PARAM=(MIN, MAX[, XAXIS])})
+ ELSE({PARAM=(MIN, MAX[, XAXIS])}) ENDIF
.SETSOA [LABEL="<STRING>"] [ANALYSIS]
+ M MNAME [SUBCKT=subckt_list|INST=inst_list] {PARAM=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="<STRING>"] [ANALYSIS]
+ M MNAME [SUBCKT=subckt_list|INST=inst_list]
+ IF(EXPR) THEN({PARAM=(MIN, MAX[, XAXIS])})
+ ELSE({PARAM=(MIN, MAX[, XAXIS])}) ENDIF
```

Set Bus Signal

```
.SIGBUS BNAME|BNAMEMSB:LSB|BNAME[MSB:LSB]|BNAME<MSB:LSB>
+ [VHI=VAL1] [VLO=VAL2] [TFALL=VAL3] [TRISE=VAL4]
+ [BASE=OCT|DEC|BIN|HEX] [SIGNED=NONE|1COMP|2COMP]
+ TN VAL {TN VAL} [P]
.SIGBUS BNAME|BNAMEMSB:LSB|BNAME[MSB:LSB]|BNAME<MSB:LSB>
+ [VHI=VAL] [VLO=VAL] [TFALL=VAL] [TRISE=VAL] [THOLD=VAL] [TDELAY=VAL]
+ [BASE=OCT|DEC|BIN|HEX] [SIGNED=NONE|1COMP|2COMP]
+ PATTERN $(PAT) {$(PAT)} |VAL {VAL} [Z]
.SIGBUS BNAME|BNAMEMSB:LSB|BNAME[MSB:LSB]|BNAME<MSB:LSB>
+ [VHI=VAL] [VLO=VAL] [TFALL=VAL] [TRISE=VAL] [THOLD=VAL] [TDELAY=VAL]
+ [BASE=OCT|DEC|BIN|HEX] [SIGNED=NONE|1COMP|2COMP]
+ FILE=FILE
```

Sinusoidal Voltage Source

```
.SINUS NODE VO VA FR [TD [THETA]]
```

Spot Noise Figure

```
.SNF INPUT=(LIST_OF_DEVICES) OUTPUT=(LIST_OF_DEVICES)
+ [INPUT_TEMP=VAL] [NOISETEMP=VAL]
```

Sizing Facility

```
.SOLVE PARAM param_name MIN MAX expr=expr [TOL=VAL]
+ [RELTOL=VAL] [GRID=VAL]
.SOLVE obj_name [W|L] MIN MAX expr=expr [TOL=VAL]
+ [RELTOL=VAL] [GRID=VAL]
.SOLVE CNAME [W|L] MIN MAX OPSIZE [TOL=VAL]
```

Simulation Start Time

```
.START_TIME time_value
```

Parameter Sweep

```
.STEP TEMP|DIPOLE INCR_SPEC
.STEP MOS W|L INCR_SPEC
.STEP MNAME PARAM_NAME INCR_SPEC
.STEP PARAM PARAM_NAME INCR_SPEC, {[VALSTART] VALSTOP VALUE}
.STEP ITEM INCR_SPEC {ITEM2 BOUND}
.STEP (ITEM1,ITEM2,...,ITEMn)
+ LIST | = (VALi1, VALi2... VALin)... (VALj1, VALj2... VALjn)
.STEP (ITEM1,ITEM2... ITEMn) LIST FILE=FILE
```

Subcircuit Definition

```
.SUBCKT NAME NN{NN}[(SWITCH|ANALOG|OSR|DIGITAL)]
+ [(NONOISE)] [(INLINE)] [PARAM: PAR=VAL {PAR=VAL}] [STATISTICAL=0|1]
...
<CIRCUIT_COMPONENTS>
...
.ENDS [NAME]
.SUBCKT LIB FNAME SNAME [LIBTYPE]
```

Subcircuit Duplicate Parameters

```
.SUBDUP SUBCKT_INST_NAME
```

Value Tables

```
.TABLE NAME (X1 Y1) {(XN YN)}
```

Set Circuit Temperature

```
.TEMP TS {TS}
```

Transfer Function

```
.TF OV IN
```

Set Title of Binary Output File

```
.TITLE name
```

Select the TOP Cell Subcircuit

```
.TOPCELL [=] <SUBCKT_NAME>
```

Transient Analysis

```
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP DATA=dataname] [UIC] [MONTE=val]
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP parameter_name
+ TYPE nb start stop] [UIC] [MONTE=val]
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP parameter_name start stop incr]
+ [UIC] [MONTE=val]
.TRAN INCRn Tn [{INCRn Tn}] [TSTART=val] [SWEEP DATA=dataname]
+ [UIC] [MONTE=val]
.TRAN INCRn Tn [{INCRn Tn}] [TSTART=val] [SWEEP parameter_name
+ TYPE nb start stop] [UIC]  [MONTE=val]
.TRAN INCRn Tn [{INCRn Tn}] [TSTART=val] [SWEEP parameter_name
+ start stop incr] [UIC] [MONTE=val]
.TRAN DATA=dataname [SWEEP DATA=dataname] [UIC] [MONTE=val]
.TRAN DATA=dataname [SWEEP parameter_name TYPE nb start stop]
+ [UIC] [MONTE=val]
.TRAN DATA=dataname [SWEEP parameter_name start stop incr]
+ [UIC] [MONTE=val]
```

Save Simulation Run at Multiple Time Points

```
.TSAVE [REPLACE|NOREPLACE] TIME=VALUE [FILE="FILEBASENAME"]
```

Test Vector Files

```
.TVINCLUDE [FILE=]FILENAME [COMP=ON|OFF] [ERRNODE[=YES|NO]]
```

Netlist Protection

```
.UNPROTECT
```

Use Previously Simulated Results

```
.USE FILE_NAME [NODESET|IC|GUESS|OVERWRITE_INPUT]
```

Use Reliability Model Key (Password)

```
.USEKEY [MODEL=model_name] KEY=key_value
```

Use Tcl File

```
.USE_TCL FILENAME [MODE=PPL|EZWAVE]
```

Test Vector Files

```
.VEC 'file_name'
```

Worst Case Analysis

```
.WCASE DC|AC|TRAN [OUTPUT=MIN|MAX|BOTH] [VARY=LOT|DEV|BOTH]
+ [TOL=VAL] [ALL] [SORT_REL=VAL] [SORT_ABS=VAL] [SORT_NBMAX=VAL]
```

Set Printer Paper Width

```
.WIDTH OUT=80|132
```

Chapter 4
Device Models

Summary

The following table summarizes the Device Models available.

Table 4-1. Eldo Device Models

Resistor	Capacitor	Inductor	Coupled Inductor
RC Wire	Diffusion Resistor	Semiconductor Resistor	
Transmission Line	Lossy Transmission Line	Lossy Transmission Line: W Model	Lossy Transmission Line: U Model
Microstrip Models			
Junction Diodes	Berkeley Level 1 (Eldo Level 1)	Modified Berkeley Level 1 (Eldo Level 2)	Fowler-Nordheim Model (Eldo Level 3)
	JUNCAP (Eldo Level 8)	Philips Diode Level 500 (Eldo Level 9)	Diode Level 21
	JUNCAP2 (Eldo Level 8, DIOLEV=11)		
BJT—Bipolar Junction Transistors	Modified Gummel-Poon Model (Eldo Level 1)	Philips Mextram 503.2 Model (Eldo Level 4)	Improved Berkeley Model (Eldo Level 5)
	VBIC v1.2 Model (Eldo Level 8)	VBIC v1.1.5 Model (Eldo Level 8)	HICUM Model (Eldo Level 9)
	Philips Mextram 504 Model (Eldo Level 22)	Philips Modella Model (Eldo Level 23)	HICUM Level0 Model (Eldo Level 24)
JFET—Junction Field Effect Transistor			
MESFET—Metal Semiconductor Field Effect Transistor			
MOSFETs	Berkeley SPICE Models	MERCKEL MOSFET Models	Berkeley SPICE BSIM1 Model (Eldo Level 8)
	Berkeley SPICE BSIM2 Model (Eldo Level 11)	Modified Berkeley Level 2 (Eldo Level 12)	Modified Berkeley Level 3 (Eldo Level 13)
	Modified Lattin-Jenkins-Grove Model (Eldo Level 16)	Enhanced Berkeley Level 2 Model (Eldo Level 17)	EKV MOS Model (Eldo Level 44 or EKV)

Table 4-1. Eldo Device Models

MOSFETs (cont.)	Berkeley SPICE BSIM3v2 Model (Eldo Level 47)	Berkeley SPICE BSIM3v3 Model (Eldo Level 53)	Motorola SSIM Model (Eldo Level 54 or SSIM)
	Berkeley SPICE BSIMSOI3 v1.3 Model (Eldo Level 55)	Berkeley SPICE BSIMSOI3 v2.x and v3.x Model (Eldo Level 56)	Philips MOS 9 Model (Eldo Level 59 or MOSP9)
	Berkeley SPICE BSIM4 Model (Eldo Level 60)	EKV3 MOS Model (Eldo Level 61 or EKV3)	TFT Polysilicon Model (Eldo Level 62)
	Philips MOS Model 11 Level 1101 (Eldo Level 63)	TFT Amorphous-Si Model (Eldo Level 64)	Philips MOS Model 11 Level 1100 (Eldo Level 65)
	HiSIM Model (Eldo Level 66)	SP Model (Eldo Level 67)	Philips MOS Model 11 Level 1102 (Eldo Level 69)
	Philips PSP Model (Eldo Level 70)	Philips MOS Model 20 Level 2002 (Eldo Level 71)	Berkeley SPICE BSIMSOI4.0 Model (Eldo Level 72)
	HiSIM-LDMOS Model (Eldo Level 73)	MOSVAR Model (Eldo Level 74)	PSP103 Model (Eldo Level 75)
	HVMOS Model (Eldo Level 101)		
S-Domain Filter	Z-Domain Filter	Subcircuit Instance	

Merging Devices in Parallel

By default, Eldo merges instances connected in parallel into a single instance. (Prior to the AMS2009.2 release this was only possible with **.OPTION REDUCE**.) The parallel instances must have the same set of parameters, and follow each other in the netlist. The multiple instances are reduced into a single instance by Eldo with the **M** parameter, for example:

```
X1 1 2 FOO A=1 B=1
X2 1 2 FOO A=1 B=1
X3 1 2 FOO A=1.0 B=1
```

Here, `X3` will remain as it is because the character string for `A` does not match, but X instances `X1` and `X2` will be replaced by:

```
X1 1 2 FOO A=1 B=1 M=2
```

This behavior applies to:

- Resistors, see “Combining Identical Resistors” on page 126,

- Diodes, see "Combining Identical Diodes" on page 227,
- BJTs, see "Combining Identical BJTs" on page 234,
- MOSFETs,see "Combining Identical MOSFETs" on page 252
- Subcircuit instances, see "Combining Identical Subcircuits" on page 303

The merging is done in order to speed-up simulations. However, this may also result in unexpected warnings in the following situations:

- If a device name is specified in a .PRINT/.PLOT/.EXTRACT, and if this device is merged with another, Eldo will issue a warning that the device is not found. For example, with .PRINT dc ix(x2.1) Eldo will issue a warning that device x2 is not found.
- Output may take into account the M factor. For example, .PRINT IX(X1.1) will return twice the value which would be obtained if devices had not been merged.
- Option MINRVAL has a different impact:

  If option MINRAVL is specified, and if there are resistors inside some X instances which are merged, the working resistance of the device is R/M, (M being the number of items in parallel. R/M can become inferior to MINRVAL while R was superior to MINRVAL. This should have limited impact on the results.

To disable merging devices in parallel, specify option **`NOCATMX`** or invoke Eldo with the **`-nocatmx`** flag.

Notes

- Monte Carlo results may vary in case devices are merged, simply because the number of random parameters to be extracted per run will not be the same. Providing that the number of Monte Carlo runs is large enough, the Monte Carlo results should be the same.
- This feature is disabled when a **`.DSPF_INCLUDE`** command is inside the netlist.
- Instances inside Verilog-A instances are never merged.

Device Models

Resistor

```
Rxx N1 N2 [MOD[EL]=MNAME] [VAL] [[TC1=]T1] [[TC2=]T2] [[TC3=]T3]
+ [AC=VAL|{EXPR}] [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [L=VAL] [W=VAL]
+ [STATISTICAL=0|1] [KEEPRMIN] [NONOISE] [NOISE=1] [KF=VAL] [AF=VAL]
+ [WEEXP=VAL] [LEEXP=VAL] [FEXP=VAL] [FMIN=VAL] [FMAX=VAL] [NBF=VAL]
Rxx N1 N2 [MOD[EL]=MNAME] [VALUE|R=]{EXPR} [ACDERFUNC=0|1]
+ [RESTORE_CAUSALITY=0|1] [[TC1=]T1] [[TC2=]T2] [[TC3=]T3] [AC=VAL]
+ [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [STATISTICAL=0|1] [KEEPRMIN] [NONOISE]
+ [NOISE=1] [KF=VAL] [AF=VAL] [WEEXP=VAL] [LEEXP=VAL] [FEXP=VAL]
+ [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [FIT=0|1] [CFMAX=VAL] [CDELF=VAL]
Rxx N1 N2 [[TC1=]T1] [[TC2=]T2] [[TC3=]T3] [AC=VAL] [T[EMP]=VAL]
+ [DTEMP=VAL] [M=VAL] [KF=VAL] [AF=VAL] [WEEXP=VAL] [LEEXP=VAL]
+ [FEXP=VAL] [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
+ TABLE EXPR [KEEPRMIN] [NONOISE] [NOISE=1]
Rxx NP NN POLY VAL {COEF} [TC1=T1] [TC2=T2] [TC3=T3] [STATISTICAL=0|1]
```

Parameters

- `xx`

  Resistor name.

- `N1`, `N2`

  Names of the resistor nodes.

- `VAL`

  Value of resistor in Ω at nominal temperature. This value can be assigned directly or via the **`.PARAM`** command. Optional if a resistor model is used. For more information, see “.PARAM” on page 778.

Note

If a parameter name is required to be specified as the 4th token of the syntax, enclose it in braces { }, single quotes ' ', or use the syntax **`r`**`=param_name` to distinguish it from the model name.

- **`MOD`**`[`**`EL`**`]=MNAME`

  Model name.

- **`TC1`**, **`TC2`**, **`TC3`**

  First, Second, and Third order temperature coefficients. Default values are zero. The temperature coefficients can be expressed simply as values, without the **`TC1`**`=` for example. **`TC3`** can be specified even if **`TC1`**/**`TC2`** are not: they would default to zero.

- **`POLY`**

  Keyword to identify the resistor as non-linear polynomial.

Note

Instead of using the **POLY** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.

- `COEF`

  Polynomial coefficients used to calculate the voltage dependency of the resistor. The value of the resistor is computed as:

  $$\text{Resistor Value} = VAL + R1 \times V + R2 \times V^2 + \ldots + Rn \times V^n$$

  where V is the voltage across the resistor.

- **AC**=VAL

  Specifies the resistor value, in Ω, which is used in AC analysis only.

- **AC={**EXPR**}**

  Enables the instantiation of a resistor with a functional expression to be simulated in AC analysis.

- **VALUE**={EXPR}**|R**={EXPR}

  Keyword indicating that the resistor has a functional description. This enables the instantiation of a frequency-dependent resistor to be simulated in all analysis modes (AC, Transient, SST and MODSST). The expression can make use of the **FREQ** keyword to specify frequency. A typical application is to model skin-effect, with **R** varying according to **SQRT(FREQ)**.

Note

If **VALUE**={EXPR} is frequency based then the allowed syntax is as follows:

Rxx NP NN **VALUE=**{EXPR} [**TC1**=T1] [**TC2**=T2] [**TC3**=T3]

VALUE={EXPR} and **AC**={EXPR} correspond to two different syntaxes, both accepted by Eldo, but they cannot be specified together.

VALUE={EXPR} can be used together with **AC**=VAL only when **VALUE**={EXPR} is *not* frequency based.

If the value of the resistor is defined by a **VALUE** expression, the actual value will be recalculated at each timestep during transient analysis. However, if an AC analysis is performed, the value is calculated only once in the DC analysis, and then considered to be static.

- **ACDERFUNC**=0|1

  Derivative of function selector for AC analysis only. For voltage or current dependent resistors, the derivatives of the value with respect to the voltage are dumped in the matrix by default. Setting **ACDERFUNC** to 0 forces Eldo to ignore these derivatives when generating the matrix. The specification on the device overrides any global setting by options **ACDERFUNC** (set by default) or **NOACDERFUNC**.

- **RESTORE_CAUSALITY**=0|1

  Having a purely real impedance that varies with the square root of frequency is not physically realizable. It would be non-causal. Thus when modeling a physical resistance, specifying a variation as the square root of frequency can be interpreted by Eldo as either:

  - specifying the module of the impedance (default, or **RESTORE_CAUSALITY**=0), or
  - specifying the real part of the impedance, and Eldo computes the imaginary part (**RESTORE_CAUSALITY**=1)

  For transient simulation, Eldo fits R with a rational function H(s), where s = j×w. In the first case, H(s) is found so that its module best fits R(f). In the second case, H(s) is found so that its real part best fits R(f).

  There is however a better way to approach frequency dependent resistor and inductance modeling, which is to specify both the real part (R(f)) and imaginary part (jw×L(f)) of impedance, with a relationship that maintains causality: this is possible using the syntax:

  ```
  Lxx N1 N2 value={expr} R value={expr}
  ```

- **T**[**EMP**]=VAL

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **DTEMP**=VAL

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **M**=VAL

  Device multiplier, simulating the effect of multiple devices in parallel: effective resistance value is divided by **M**. Default is 1.

  The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**, **MAXL**, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note

Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- **KEEPRMIN**

  Any effect of **RMMINRVAL** and **MINRVAL** options will be ignored for the device. This is a way to force Eldo to keep the device under all circumstances unless the device is equal to zero.

- **NONOISE**

  Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the device presents no noise contribution to the noise analysis.

- **NOISE**=1

  Specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **NONOISE** specification on a **.MODEL** card.

- **TABLE**

  Keyword indicating that the resistor accepts a table description.

- **KF**=VAL

  Flicker noise coefficient. Default is 0.

- **AF**=VAL

  Flicker noise exponent. Default is 1.0.

- **WEEXP**=VAL

  Flicker noise exponent. Default is 0.0.

- **LEEXP**=VAL

  Flicker noise exponent. Default is 0.0.

- **FEXP**=VAL

  Flicker noise exponent. Default is 1.0. Flicker noise equation is:

$$Sid = KF \cdot \frac{I^{AF}}{W_{eff}^{WEEXP} \cdot L_{eff}^{LEEXP} \cdot freq^{FEXP}}$$

- **FMIN**=VAL

  Lower limit of the noise frequency band.

- **FMAX**=VAL

  Upper limit of the noise frequency band.

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion. **FMIN** is also used to specify the algorithm used to generate the noise source generated by the resistor. When **FMIN**>0 the resistor noise source is generated with sinusoids; when **FMIN**=0 it is generated with a continuous spectrum between **FMIN** and **FMAX**.

- **NBF**=VAL

  Specifies the number of sinusoidal sources with appropriate amplitude and frequency and with randomly distributed phase from which the noise source is composed. Default value is 50. This parameter has no effect when **FMIN** is set to 0.

- **FIT**=0|1

  Specifies the method used to perform transient analysis. **FIT**=1 indicates the fitting method is used, **FIT**=0 indicates the convolution method is used. Default value is 1.

- **CFMAX**=VAL

  Specifies the maximum frequency value used in the functional description of the resistor. It is used in transient analysis to perform impulse response using a convolution method. Default is 1.0×10^9 Hz. Can only be used when **FIT**=0.

- **CDELF**=VAL

  Specifies the frequency interval to compute the functional description of the resistor. It must have the form 2^n for the convolution computation (automatic check and correction are done if not). Default is $1.0\times10^9/1024 = 9.765625\times10^5$ Hz. Can only be used when **FIT**=0.

Note

The parameters **FIT**, **CFMAX**, **CDELF** should only be used when the resistor is frequency dependent, else they will be ignored.

The user can specify the minimum value that resistors in the netlist can take using the option **RSMALL**. For more information please refer to “RSMALL=VAL” on page 990.

Combining Identical Resistors

Multiple identical resistors connected in parallel are reduced into a single instance using the **M** parameter, for example:

```
r1 1 2 res1
r2 1 2 res1
r3 1 3 res1
```

Here, r3 will remain as it is because it is connected to different nodes, but resistor instances r1 and r2 will be replaced by:

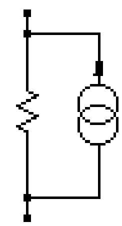
```
r1 1 2 res1 M = 2
```

Note

It will only work when no parameters are given on the instance.

For more information see "Merging Devices in Parallel" on page 120.

Noise in Resistors



The thermal noise of a resistor is as follows:

$$SI = \frac{4 \times k \times T}{R}$$

Examples

```
r1 n3 n4 3.3k
```

Specifies a 3.3kΩ resistor placed between nodes n3 and n4.

```
r2 n1 n2 rval
.param rval=2k
```

Specifies a resistor named r2 of value rval between nodes n1 and n2. The resistor value is declared globally in the `.PARAM` command.

```
r3 1 2 value={2k*v(3,4)*i(v5)}
```

Specifies a resistor r3 between nodes 1 and 2 whose value is described by the expression in curly brackets.

```
rg4 4 5 table (v(p3n)) = (0, 1e11) (1v, 1e3)
```

The value of resistor rg4 is 1×10^{11} Ω when v(p3n)=0V, 1kΩ when v(p3n)=1V. The other values are interpolated.

```
r5 1 2 value={50*sqrt(1+(FREQ/10e6))}
```

Specifies a frequency-dependent resistor r5 between nodes 1 and 2 whose value is described by the expression in curly brackets.

```
r1 1 2 ac=3 value={2k*v(3,4)*i(v5)}
```

Specifies a resistor r1 between nodes 1 and 2 whose value is described by the expression in curly brackets. This value is superseded when an AC analysis is performed, where the ac value is used instead, in this case 3Ω.

```
r2 3 4 value={50*sqrt(1+(FREQ/10e6))} tc1=0.001
+ tc2=0.004 tc3=0.003
```

Specifies a frequency dependent resistor r2 between nodes 3 and 4, whose value is described by the expression in curly brackets, that also has first, second and third order temperature coefficients.

```
r1 1 2 value = {1k + 2k*v(1)} ACDERFUNC = 1
```

Specifies a voltage dependent resistor. The derivatives of the value with respect to the voltage are dumped in the matrix because **ACDERFUNC** (derivative of function selector) is set to 1, therefore the term 2k will be in the matrix. By default (without **ACDERFUNC**) it will not be in the matrix.

Resistor Model Syntax

```
.MODEL MNAME R[ES] [{PAR=VAL}]
```

Note

Specifying `SCALM=val` in the `.MODEL` statement in a resistor model overrides the global scaling specified by the `.OPTION SCALM=val` statement.

Level 1

Table 4-2. Resistor Model—Level 1 Parameters

Nr.	Name	Description	Default	Units
1	R or SCALE[R]	Resistance multiplier	1	
2	RDEF	Value of resistor. This value has priority over values computed from L and W	1×10^3	Ω
3	POLY[1]	Keyword to identify the resistor as non-linear polynomial		
4	POLYV	Used to change the formula used for computing the R value	2	
5	REVSP	Used to change the formula used for computing the R value	0	

Table 4-2. Resistor Model—Level 1 Parameters

Nr.	Name	Description	Default	Units
6	TC1	First order temperature coefficient	0	$°C^{-1}$
7	TC2	Second order temperature coefficient	0	$°C^{-2}$
8	TC3	Third order temperature coefficient	0	$°C^{-3}$
9	LOT	Correlated device tolerance (Monte Carlo analysis)		
10	DEV	Uncorrelated device tolerance (Monte Carlo analysis)		
11	NOISE	Noise contribution of resistor[2]	1.0	
12	WEEXP	Flicker noise exponents	0.0	
13	LEEXP		0.0	
14	FEXP		1.0	
15	TNOM	Nominal temperature	27	°C

1. Instead of using the **POLY** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.
2. Noise contribution of the resistor will be expressed as 4×***KB***×**T**×**NOISE**/**R**. Where ***KB*** is the Boltzmann coefficient.

The below equation defines resistor value as a function of temperature, where T is the operating temperature specified either by the **.TEMP** command, or the **T** parameter. Tnom is the nominal temperature for which the resistor has resistance VAL. Default value of Tnom is 27 °C and is adjustable using **.OPTION TNOM**.

$$RVAL(T) = VAL(T_{nom})(1 + \mathrm{TC1}(T - T_{nom}) + \mathrm{TC2}(T - T_{nom})^2 + \mathrm{TC3}(T - T_{nom})^3)$$

The Resistor model has additional levels for the **.MODEL** card.

Note

TC1, TC2, TC3 and R are used by model levels 1, 2, 3 and 4.

POLY usage

The **POLY** keyword identifies the model as a non-linear polynomial. It applies to the resistance/capacitance/inductor model syntax and is accepted on the model card in which the coefficients are used to generate R, C or L values which are dependent on bias. Syntax:

```
.MODEL <model_name> [R|C|IND] POLY <list_of_parameter_values>
```

This polynomial list can be given in addition to existing model parameters, for example:

```
.MODEL <model_name> R TC1=0.1 POLY <list_of_parameter_values>
+ TC2=0.2
```

The bias independent value of the resistor will be computed as if there were no polynomial parameters. Then, the active value of the resistor will be equal to the product of the “bias-independent value” multiplied by the polynomial expression.

Notes:

- **`POLY`** can be specified in a capacitance instance the same way it is used for resistance.
- Instead of using the **`POLY`** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.

REVSP and POLYV usage

- Parameter **`POLYV`** can also be specified on the **`.MODEL`** card. **`POLYV`** can take two values: equal to 2 or not equal to 2. It will be used to change the formula which is used for computing the R/L/C value. Assuming the model:

  ```
  .MODEL Foo C POLY P1 P2 P3... Pn
  C pin1 pin2 Foo <cinst>
  ```

  If V = V(pin1) - V(pin2):

  then if **`POLYV`** is 2: the value of C will be computed as:

  $$C = \langle cinst \rangle \times (P1 + P2 \times V + P3 \times V^2 + ...)$$

  if **`POLYV`** is not 2: the value of C will be computed as:

  $$C = \langle cinst \rangle \times (1 + P1 \times V + P2 \times V^2 + ...)$$

  P1and P2 represent the polynomial coefficients. The default value for **`POLYV`** is 2.

- Parameter flag **`REVSP`** can also take two values: equal to 1 or not equal to 1. **`POLYV`** and **`REVSP`** parameters are flags to modify computations. They do not interact directly within the computation. If parameter **`REVSP`** is set to 1 on the **`.MODEL`** card, and if **`POLYV`** is not 2, then the formula for computing the current generated by the resistor is:

  $$I = \frac{V}{rinst} \times \left(1 + \frac{P1 \times V}{2} + \frac{P2 \times V^2}{3} + \ldots\right)$$

  which leads to a resistor of value:

  $$R(V) = \frac{1}{\delta I / \delta V} = \frac{rinst}{(1 + P1 \times V + P2 \times V^2 + \ldots)}$$

  P1and P2 represent the polynomial coefficients. The default value for **`REVSP`** is 0.

Example

```
.model rmodel res tc1=0.001 tc2=0.005 tc3=0.008
...
r2 n1 n19 rmodel 2.5k
```

Specifies a 2.5kΩ resistor placed between nodes `n1` and `n19`. The first order temperature coefficient is 0.001, the second order temperature coefficient is 0.005, and the third order temperature coefficient is 0.008, all being defined using the **`.MODEL`** command.

Level 2

This is a private ST model. For a description of equations please contact STMicroelectronics. This is the default model when the `-stver` flag is specified (or option **`stver`**). Parameters:

Table 4-3. Resistor Model—Level 2 Parameters

Nr.	Name	Description	Default	Units
1	RHO	Sheet resistance	0.0	Ω/sq
2	RCON[1]	Resistance of contacts	0	Ω
3	NC1 (NC)	Number of ohmic contacts at node 1		
4	NC2	Number of ohmic contacts at node 2		
5	DL	Delta length	0	m
6	DW	Delta width	0	m
7	POLY	Keyword to identify the resistor as non-linear polynomial		

1. RCON is taken into account in the Resistor computation if NC1, NC2 are defined.

Level 3

This corresponds to the RC Wire model.

Please see the "RC Wire" on page 148.

Level 4

This is the default model inside Accusim. Parameters:

Table 4-4. Resistor Model—Level 4 Parameters

Nor.	Name	Description	Default	Units
1	RSH	Sheet resistance	50.0	Ω/sq.
2	LNARROW	Delta length	0	m
3	WNARROW	Delta width	0	m

Table 4-4. Resistor Model—Level 4 Parameters

Nor.	Name	Description	Default	Units
4	NARROW	Delta	0	m
5	R	Multiplier (used for Monte Carlo)	1	

The device value is computed as follows:

$$R = RSH \times \frac{Leff}{Weff}$$

where:

- L_{eff} = L - LNARROW if **LNARROW** is specified, or
  L_{eff} = L - NARROW if **NARROW** is specified.
- W_{eff} = W - WNARROW if **WNARROW** is specified, or
  W_{eff} = W - NARROW if **NARROW** is specified.

Level 6

This corresponds to the Diffusion resistor model.

Please see the “Diffusion Resistor” on page 154.

Capacitor

```
Cxx NP NN [MOD[EL]=MNAME] [DCCUT] [VAL] [M=VAL] [L=VAL] [W=VAL]
+ [T[EMP]=VAL] [DTEMP=VAL] [TC1=T1] [TC2=T2] [TC3=T3] [IC=VAL]
+ [STATISTICAL=0|1]
Cxx NP NN POLY VAL {COEF} [TC1=T1] [TC2=T2] [TC3=T3] [M=VAL]
+ [CTYPE=VAL] [IC=VAL] [STATISTICAL=0|1]
Cxx NP NN [VALUE=|C=]{EXPR} [ACDERFUNC=0|1] [RESTORE_CAUSALITY=0|1]
+ [TC1=T1] [TC2=T2] [TC3=T3] [CTYPE=VAL] [STATISTICAL=0|1]
```

Parameters

- `xx`

  Capacitor name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `VAL`

  Value of the capacitor in Farads (voltage independent value). This value can be assigned directly or via the **`.PARAM`** command. Optional if a capacitor model is used. For more information, see “.PARAM” on page 778.

- **`POLY`**

  Keyword to identify the capacitor as non-linear polynomial.

Note

Instead of using the **`POLY`** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.

- **`VALUE`**`={EXPR}|`**`C`**`={EXPR}`

  Keyword indicating that the capacitor has a functional description. This enables the instantiation of a frequency-dependent capacitor to be simulated in all analysis modes (AC, Transient, SST and MODSST). The expression can make use of the FREQ keyword to specify frequency. A typical application is to model skin-effect, with C varying according to SQRT(FREQ).

- **`ACDERFUNC`**`=0|1`

  Derivative of function selector for AC analysis only. For voltage or current dependent capacitors, the derivatives of the value with respect to the voltage are dumped in the matrix by default. Setting **`ACDERFUNC`** to 0 forces Eldo to ignore these derivatives when generating the matrix. The specification on the device overrides any global setting by options **`ACDERFUNC`** (set by default) or **`NOACDERFUNC`**.

- **RESTORE_CAUSALITY**=0|1

  Having a purely real impedance that varies with the square root of frequency is not physically realizable. It would be non-causal. Thus when modeling a physical resistance, specifying a variation as the square root of frequency can be interpreted by Eldo as either:

  - specifying the module of the impedance (default, or **RESTORE_CAUSALITY**=0), or
  - specifying the real part of the impedance, and Eldo computes the imaginary part (**RESTORE_CAUSALITY**=1)

- MNAME

  Model name. This is useful in combination with Monte Carlo analysis.

- **DCCUT**

  For DC, TRAN, MODSST analyses, the **DCCUT** device is assumed to be a normal capacitance of VAL Farads. For **.AC**, **.SST**, **.SSTAC**, **.SSTNOISE** and **.SSTXF** analyses, the **DCCUT** device corresponds to an open circuit in DC and a short circuit for all other frequencies.

- COEF

  Polynomial coefficients used to calculate the voltage dependency of the capacitance. The value of the capacitor is computed as:

  $$\text{Capacitor Value} = VAL + C1 \times V + C2 \times V^2 + \ldots + Cn \times V^n$$

  where V is the voltage across the capacitor.

- M=VAL

  Multiplier representing the number of parallel devices in the simulation. The total capacitor value will be multiplied by this parameter value. Default value=1.0.

  The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**, **MAXL**, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note

Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- L=VAL

  Capacitor length (Effective L = ***L***×***SHRINK***), see model parameter list where the shrink factor is listed.

- `W=VAL`

  Capacitor width (Effective W = ***W***×***SHRINK***), see model parameter list where the shrink factor is listed.

- `**T**[**EMP**]=VAL`

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **`DTEMP`**`=VAL`

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

`TEMP` and **`DTEMP`** are mutually exclusive. If both are specified, the last one is used.

- **`TC1`**, **`TC2`**, **`TC3`**

  First, Second, and Third order temperature coefficients. Default values are zero.

- **`IC`**`=VAL`

  Sets the initial guess for the voltage across the capacitor prior to a transient analysis. To use this option, the **`UIC`** parameter must also be present in the **`.TRAN`** statement. For more information, see “.TRAN” on page 911.

- **`STATISTICAL`**`=0|1`

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **`STATISTICAL`**. Default is 1.

- **`CTYPE`**`=VAL`

  Determines the calculation mode for capacitors defined by polynomial or functional expressions (**`VALUE`**`={EXPR}`). The default value is 0.

  When **`CTYPE`**=0, the current out of the bias-dependent capacitor is computed as I(C) = dQ/dt, with Q computed by integrating the equation dQ = C×dV. Default.

  When **`CTYPE`**=1, the current out of the bias-dependent capacitor is also computed as I(C) = dQ/dt, but with Q computed as Q = C×V.

  When **`CTYPE`**=2, a charge equation is used, the current out of the bias-dependent capacitor is computed as I(C) = dQ/dt, with Q provided by the expression and not computed.

  Eldo will check dependencies of the capacitor value. When the value of the capacitor does not depend on the bias of the pins of the device, the **`CTYPE`**=1 formulation is normally more appropriate. In this case Eldo will behave on the device as if **`CTYPE`**=1 had been set, unless option **`NOAUTOCTYPE`** is set, or unless **`CTYPE`** is explicitly set on the device.

Note

The IC=VAL statement cannot be given immediately after a model name, for instance in the following example: **Cxx NP NN MNAME IC=value**, MNAME would not be considered as a model name, but as the value of the capacitor, and Eldo would look for a parameter named MNAME. Ensure any other parameter is inserted preceding the IC parameter, for example: **Cxx NP NN MNAME TC1=0 IC=value**

Note

If the value of the capacitor is defined by a **POLY** or **VALUE** expression, the actual value will be recalculated at each timestep during transient analysis. If an AC analysis is performed however, the value is calculated only once in the DC analysis, and then considered to be static.

Examples

```
c1 n3 n4 0.5pf
```

Specifies a 0.5pF capacitor c1 placed between nodes n3 and n4.

```
c1 n3 n7 poly 5p 0.1p 0.07p 0.004p
```

Specifies a 5pF capacitor c1 placed between nodes n3 and n7, whose voltage dependency is described by the 3rd order polynomial:

$$value = 5p + 0.1p \times V + 0.07p \times V^2 + 0.004p \times V^3$$

where V is the voltage across the capacitor.

```
c2 n1 n2 cval
.param cval=0.4p
```

Specifies a capacitor c2 of value cval placed between nodes n1 and n2. The capacitor value is declared globally in the **.PARAM** command.

```
c1 1 2 value={2n*v(3, 4)*i(v5)}
```

Specifies a capacitor c1 between nodes 1 and 2 whose value is described by the expression in curly brackets.

Dynamic current calculation (**CTYPE**) example:

```
v10 2 0 sin (0 1 100meg)
r10 2 0 1
v1 1 0 1
c1 1 0 value = {1.0e-9 * v(2,0)}
.tran 1n 100n
.print tran i(v1)
.plot tran i(v1)
.end
```

Eldo will issue a warning that **CTYPE**=1 is emulated on device C1.

Capacitor Model Syntax

```
.MODEL MNAME C[AP] [{PAR=VAL}]
```

The Eldo capacitor model supports Monte Carlo analysis whereby the tolerance of the capacitor can be defined to vary in a correlated or un-correlated way over a number of simulation runs.

Note

Specifying **SCALM**=val in the **.MODEL** statement in a capacitor model overrides the global scaling specified by the **.OPTION SCALM**=val statement.

Table 4-5. Capacitor Model Parameters

Nr.	Name	Description	Default	Units
1	C[1] or SCALE[C]	Capacitance multiplier	1	
2	CDEF	Value of capacitor if value isn't given in instance	0	F
3	LOT	Correlated device tolerance (Monte Carlo analysis)		
4	DEV	Uncorrelated device tolerance (Monte Carlo analysis)		
5	POLY[2]	Keyword to identify the capacitor as non-linear polynomial		
6	POLYV	Used to change the formula used for computing the C value	2	
7	REVSP	Used to change the formula used for computing the C value	0	
8	TC1	First order temperature coefficient	0	$°C^{-1}$
9	TC2	Second order temperature coefficient	0	$°C^{-2}$
10	TC3	Third order temperature coefficient	0	$°C^{-3}$
11	CAPSW	Sidewall fringing capacitance	0	Fm^{-1}
12	COX[3]	Bottomwall capacitance	0	Fm^{-2}
13	DEL	Small decrement in dimensions of a device structure on both ends of L and W due to process effects (undercutting)	0	
14	DI	Relative dielectric constant	3.9	
15	SHRINK[4]	Shrink factor for physical dimensions	1	
16	THICK	Insulator thickness	0	m
17	L	Default capacitor length	0	m

Table 4-5. Capacitor Model Parameters

Nr.	Name	Description	Default	Units
18	W	Default capacitor width	0	m
19	DTEMP	Temperature difference of capacitor with respect to the rest of the circuit	0	
20	SCALM	Model parameter scaling factor. This overrides the global **SCALM** value defined using the **.OPTION** command.	1	
21	TNOM	Nominal temperature	27	°C

1. Is computed as $L_{eff} \times W_{eff} \times C_{oxeff} + 2 \times (L_{eff} + W_{eff}) \times CAPSW$
2. Instead of using the **POLY** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.
3. C_{oxeff} is computed as $EPSO \times DI / TOX$ if ***TOX*** is specified, $EPSO \times DI / THICK$ if ***THICK*** is specified (***TOX*** and ***THICK*** are synonymous), $EPSO \times DI / 0.5\mu$ otherwise.
4. Effective value of length = $(L \times SHRINK - 2 \times DEL \times SCALEM)$
Effective value of width = $(W \times SHRINK - 2 \times DEL \times SCALEM)$

If **COX** is not specified, it is computed as:

$$COX = \frac{\varepsilon o \times DI}{THICK} \text{ if } \boldsymbol{DI} \text{ is specified, } COX = \frac{\varepsilon x}{THICK} \text{ otherwise,}$$

where εo is the permittivity of air, and εox is the permittivity of the oxide.

Tip: For detailed information on usage of **POLY**, **REVSP**, and **POLYV** parameters, see "POLY usage" on page 129.

Examples

```
*MODEL definition
.model cmodel cap lot=2%
...
*main circuit
c3 1 s cmodel 0.5pf
```

Specifies a 0.5pF capacitor c3 placed between the nodes 1 and s, using the **.MODEL** command to define its parameters for a Monte Carlo analysis.

```
c1 1 2 cmodel1 tc1=25e-3 m=2 dtemp=10
.model cmodel1 c cox=1e-12 capsw=1e-12 del=0.01
+ l=1 w=1 tc1=50e-6 tc2=0
```

These are valid instance and model cards from which nominal capacitance may be evaluated from a simplified equation (no **SHRINK** or scale factors):

Capacitance =
$$\{(L - 2 \times DEL) \times (W - 2 \times DEL) \times COX + 2 \times (L - 2 \times DEL) \times (W - 2 \times DEL) \times CAPSW\}$$

In the following example, the bias independent value of the capacitor will be computed as if there were no **POLY** parameters. Then, the active value of the capacitor will be equal to the product of the "bias-independent-value" multiplied by the polynomial expression.

```
.MODEL model_name C TC1=1 POLY
+ <list_of_parameter_values> TC2=value
```

POLYV example:

Assuming the model:

```
.MODEL Foo C POLY P1 P2 P3.. Pn
C pin1 pin2 Foo <cinst>
```

And, for example, $V = V(pin1) - V(pin2)$:

- If **POLYV**=2: the value of *C* will be computed as:
  $C = <cinst>*(P1 + P2*V + P3*V*V + ...)$
- If **POLYV**≠2: the value of *C* will be computed as:
  $C = <cinst>*(1 + P1*V + P2*V*V + ...)$

Inductor

```
Lxx NP NN [MOD[EL]=MNAME] [DCFEED] [VAL] [M=VAL1] [T[EMP]=VAL] [DTEMP=VAL]
+ [IC=VAL3] [TC1=T1] [TC2=T2] [TC3=T3] [R=VAL4] [STATISTICAL=0|1]
Lxx NP NN POLY VAL {LN} [IC=VAL] [R=VAL] [TC1=T1] [TC2=T2] [TC3=T3]
+ [STATISTICAL=0|1]
Lxx NP NN [VALUE=|L=]{EXPR} [ACDERFUNC=0|1] [RESTORE_CAUSALITY=0|1]
+ [R=VAL|R VALUE=EXPR|R TABLE {fval rval}]
+ [TC1=T1] [TC2=T2] [TC3=T3]  [LTYPE=VAL] [STATISTICAL=0|1]
Lxx {port_list} KMATRIX=data_block [STATISTICAL=0|1]
Lxx {port_list} RELUCTANCE=({rn,cn,valn}) <options> [STATISTICAL=0|1]
Lxx {port_list} RELUCTANCE {FILE="file"} <options> [STATISTICAL=0|1]
```

Parameters

- `xx`

  Inductor name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `VAL`

  Inductor value in Henrys. This value can be assigned directly or via the **`.PARAM`** command. Optional if an inductor model is used. For more information, see “.PARAM” on page 778.

- **`POLY`**

  Keyword to identify the inductor as non-linear polynomial.

Note

Instead of using the **`POLY`** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.

- **`VALUE`**`={EXPR}|`**`L`**`={EXPR}`

  Keyword indicating that the inductor has a functional description. This enables the instantiation of a frequency-dependent inductor to be simulated in all analysis modes (AC, Transient, SST and MODSST). The expression can make use of the FREQ keyword to specify frequency. A typical application is to model skin-effect, with L varying according to SQRT(FREQ).

- **`ACDERFUNC`**`=0|1`

  Derivative of function selector for AC analysis only. For voltage or current dependent inductors, the derivatives of the value with respect to the voltage are dumped in the matrix by default. Setting **`ACDERFUNC`** to 0 forces Eldo to ignore these derivatives when generating the matrix. The specification on the device overrides any global setting by options **`ACDERFUNC`** (set by default) or **`NOACDERFUNC`**.

- **RESTORE_CAUSALITY**=0|1

  Having a purely real impedance that varies with the square root of frequency is not physically realizable. It would be non-causal. Thus when modeling a physical resistance, specifying a variation as the square root of frequency can be interpreted by Eldo as either:

  - specifying the module of the impedance (default, or **RESTORE_CAUSALITY**=0), or
  - specifying the real part of the impedance, and Eldo computes the imaginary part (**RESTORE_CAUSALITY**=1)

  For transient simulation, Eldo fits R with a rational function H(s), where s = j×w. In the first case, H(s) is found so that its module best fits R(f). In the second case, H(s) is found so that its real part best fits R(f).

  There is however a better way to approach frequency dependent resistor and inductance modeling, which is to specify both the real part (R(f)) and imaginary part (jw×L(f)) of impedance, with a relationship that maintains causality: this is possible using the syntax:

  ```
  Lxx N1 N2 value={expr} R value={expr}
  ```

- MNAME

  Model name. This is useful in combination with Monte Carlo analysis. If **MOD=** is specified, Eldo will look for a **.MODEL** card. If not specified, Eldo will look for a parameter named **foo**.

- **DCFEED**

  For DC, TRAN, MODSST analyses, the **DCFEED** device is assumed to be a normal inductance of VAL Henrys. For **.AC**, **.SST**, **.SSTAC**, **.SSTNOISE** and **.SSTXF** analyses, the **DCFEED** device corresponds to a short circuit in DC and an open circuit for all other frequencies.

- LN

  Coefficients of a polynomial used to calculate the inductances. The inductor is expressed as a function of the current I across the element. The value of the inductance is computed as:

  $$\text{Inductor Value} = VAL + \text{L1} \times I + \text{L2} \times I^2 + \ldots + \text{Ln} \times I^n$$

  where I is the current through the inductor.

- **IC**=VAL

  Sets the initial ‘guess’ for the current through the inductor prior to a transient analysis. To use this option the **UIC** parameter must also be present in the **.TRAN** statement.

- M=VAL

  Multiplier representing the number of parallel devices in the simulation. Default value=1.0.

  The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**,

MAXL, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note
Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **T**[**EMP**]=VAL

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **DTEMP**=VAL

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note
TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **TC1**, **TC2**, **TC3**

  First, Second, and Third order temperature coefficients.

- R=VAL|**R VALUE**={EXPR}|**R TABLE**={fval rval}

  R is a resistor that is added in series with inductor **L**. Default value is 0.0. When **R VALUE**={EXPR} is specified, it indicates that the resistor has a functional description, enabling a frequency-dependent resistor. The expression can make use of the **FREQ** keyword to specify frequency. When **R TABLE** is specified, it indicates that the resistor accepts a table description, with pairs of frequency and resistor values specified. This device is supported for all analyses (DC, AC, TRAN, SST, MODSST, and so on).

Note
If the value of the inductor is defined by a **POLY** or **VALUE** expression, the actual value will be recalculated at each timestep during transient analysis. If an AC analysis is performed however, the value is calculated only once in the DC analysis, and then considered to be static.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- **LTYPE**=VAL

  Determines the calculation mode for inductors defined by functional expressions (**VALUE**={EXPR}). The default value is 0.

When **LTYPE**=0, the voltage across the inductor is computed as V = dFlux/dt, with Flux computed by integrating the equation dFlux = I×dV. Default.

When **LTYPE**=1, the voltage across the inductor is also computed as V = dFlux/dt, but with Flux computed as Flux = I×V.

When **LTYPE**=2, a flux equation is used, the voltage across the inductor is computed as V = dFlux/dt, with Flux provided by the expression and not computed.

- **KMATRIX**=data_block

  K-element for modeling the parasitic inductive effect of general 3-D interconnects. It is described by a multi-port inductor record and the matrix values are provided through a data block (**.DATA**). Values in the matrix are reluctance values, not inductance values.

  Eldo will check that:

  - the K-element has an even number of nodes
  - the **KMATRIX** has a valid value and a data block exists
  - the matrix is positive definite
  - all diagonal terms are defined and have positive values
  - there is no duplicate entry and no term is defined below the diagonal

- port_list

  List of ports for K-element, specified as pairs of electrical node names, each defining an interconnect branch.

- **RELUCTANCE**

  Keyword indicates that reluctance (inverse inductance) will be provided. The unit for reluctance is inverse Henry (H^{-1}). The presence of this keyword indicates that all tokens between Lname and the RELUCTANCE keyword should be interpreted as node names.

- **RELUCTANCE**=({rn,cn,valn})

  Inline form of reluctance specification for modeling the parasitic inductive effect of general 3-D interconnects. rn,cn,valn is a triplet of values describing the reluctance between two interconnect branches.

  rn integer value of branch #1; integer refers to the pair of nodes that define branch #1

  cn integer value of branch #2; integer refers to the pair of nodes that define branch #2

  valn reluctance value between branch #1 and branch #2 (H^{-1})

- **FILE**=”file”

  External file format of reluctance specification for modeling the parasitic inductive effect of general 3-D interconnects. The data files should contain three columns of data. Each row should contain an (r, c, val) triplet separated by spaces. The r, c, and val values may be expressions surrounded by single quotes. Multiple files may be specified to allow the reluctance data to be spread over several files if necessary. The files should not contain a header row.

- `<options>`

  Valid options for reluctance specification include:

  SHORTALL=yes|no

  causes all inductors to be converted to short circuits, and all reluctance matrix values to be ignored.

  IGNORE_COUPLING=yes|no

  causes all off-diagonal terms to be ignored (i.e., set to zero).

Examples

```
l1 n13 n8 5u
```

Specifies a 5µH inductor l1 placed between nodes n13 and n8.

```
l1 n10 n5 poly 7u 0.15 0.03
```

Specifies a 7µH inductor l1 placed between nodes n10 and n5, whose current dependency is described by a second order polynomial of the form:

$$value = 7\mu + 0.15 \times I + 0.003 \times I^2$$

where I is the current across the inductor.

```
l3 n1 n2 lval
.param lval=0.6ph
```

Specifies an inductor named l3 of value lval placed between nodes n1 and n2. The inductor value is declared globally in the **.PARAM** command.

```
l1 1 2 value={2u*v(3, 4)*i(v5)}
```

Specifies an inductor l1 between nodes 1 and 2 whose value is described by the expression in curly brackets.

K-element example:

```
L_ThreeNets (a,1) (1,2) (2,a_1) (b,4) (4,5) (5,b_1) (c,7) (7,8)
(8,c_1) KMATRIX=StartK
.DATA StartX
+ PORT1 PORT2 VALUE
+ 1     1     103e9
+ 1     4     -34.7e9
+ 1     7     -9.95e9
+ 4     4     114e9
+ 4     7     -34.7e9
+ 7     7     103e9
+ 2     2     103e9
+ 2     5     -34.7e9
+ 2     8     -9.95e9
+ 5     5     114e9
+ 5     8     -34.7e9
```

```
+ 8     8     103e9
+ 3     3     103e9
+ 3     6     -34.7e9
+ 3     9     -9.95e9
+ 6     6     114e9
+ 6     9     -34.7
+ 9     9     103e9
```

Another K-element example using the alternate inline syntax:

```
L_ThreeNets a 1 1 2 2 a_1 b 4 4 5 5 b_1 c 7 7 8 8 c_1
+ RELUCTANCE=(
+ 1, 1, 103e9,
+ 1, 4, -34.7e9,
+ 1, 7, -9.95e9,
+ 4, 4, 114e9,
+ 4, 7, -34.7e9,
+ 7, 7, 103e9,
+ 2, 2, 103e9,
+ 2, 5, -34.7e9,
+ 2, 8, -9.95e9,
+ 5, 5, 114e9,
+ 5, 8, -34.7e9,
+ 8, 8, 103e9,
+ 3, 3, 103e9,
+ 3, 6, -34.7e9,
+ 3, 9, -9.95e9,
+ 6, 6, 114e9,
+ 6, 9, -34.7e9,
+ 9, 9, 103e9 )
```

Another K-element example using the alternate external file syntax:

```
L_ThreeNets a 1 1 2 2 a_1 b 4 4 5 5 b_1 c 7 7 8 8 c_1 RELUCTANCE
+ FILE=”reluctance.dat”
```

where the file *reluctance.dat* contains the three columns of data.

Inductor Model Syntax

```
.MODEL MNAME IND [{PAR=VAL}]
```

The inductor model provided with Eldo is, as with the capacitor model described before, used in conjunction with a Monte Carlo analysis whereby the tolerance of the inductor can be defined to vary in a correlated or uncorrelated way during a number of simulation runs.

Table 4-6. Inductor Model Parameters

Nr.	Name	Description	Default	Units
1	L or SCALEL	Inductance multiplier	1	
2	LOT	Correlated device tolerance (Monte Carlo analysis)		
3	DEV	Uncorrelated device tolerance (Monte Carlo analysis)		

Table 4-6. Inductor Model Parameters

Nr.	Name	Description	Default	Units
4	POLY[1]	Keyword to identify the inductor as non-linear polynomial		
5	POLYV	Used to change the formula used for computing the L value	2	
6	REVSP	Used to change the formula used for computing the L value	0	
7	TC1	First order temperature coefficient	0	$°C^{-1}$
8	TC2	Second order temperature coefficient	0	$°C^{-2}$
9	LDEF	Inductor value to be provided to the inductor instance if no instance parameters are specified	0	H
10	TNOM	Nominal temperature	27	°C

1. Instead of using the **POLY** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.

Tip: For detailed information on usage of **POLY**, **REVSP**, and **POLYV** parameters, see "POLY usage" on page 129.

Examples

The following specifies a 0.5 pH inductor placed between the nodes 1 and s, using the **.MODEL** command to define the parameters for a Monte Carlo analysis.

```
*MODEL definition
.model lmodel ind lot=2%
...
*main circuit
l1 1 s lmodel 0.5ph
```

In the following, the **scalel** parameter multiplies the inductance value given in the instantiation. Therefore this inductor will have a value of 6 pH.

```
*MODEL definition
.model lmodel ind scalel=3
...
*main circuit
l1 1 s lmodel 2ph
```

In the following, the value of L1 will be 1uH. This shows how the inductor **.MODEL** card provides the value to the inductor instance when no instance parameters are specified,

```
.MODEL foo IND LDEF=1u
L1 1 2 MOD=foo
```

Coupled Inductor

```
Kxx Lyy Lzz KVAL [KR=KRVAL]
```

Parameters

- `xx`

  Name of the coupled inductor.

- `yy, zz`

  Names of the two inductors building **K**xx.

- `KVAL`

  Coupling coefficient where usual value is $-1 \le KVAL \le +1$. If a value outside this range is specified, Eldo will generate the warning message:

  ```
  unusual coefficient ... <KVAL>
  ```

- `KR=KRVAL`

  KR is a resistor that is coupled with the coupled inductor **K**. Default value is 0.0.

  Mutual inductance is given by: $Mxx = KVAL \times \sqrt{Lyy \times Lzz}$

  Mutual resistance is given by: $Rxx = KRVAL \times \sqrt{Ryy \times Rzz}$

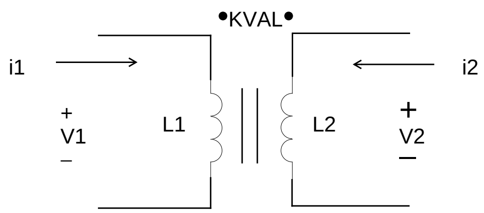
  where:

  Lyy, *Lzz*, *Ryy*, *Rzz* are the inductance and resistance values of the corresponding impedance.

  Usually $-1 \le KVAL \le +1$, otherwise a warning is generated.

  When a coupling resistance coefficient *KRVAL* is specified on the K statement, R=Rval must be specified on the corresponding impedance statements, otherwise *KRVAL* is ignored.

Figure 4-1. Coupled Inductor



Example

```
l7 4 3 0.7m
l8 5 9 0.4m
k12 l7 l8 0.2
```

Specifies the coupling of inductors l7 and l8, with a coupling coefficient of 0.2. The coupling element, or transformer, is given the symbol k12.

RC Wire

```
Rxx N1 N2 MNAME [[R=]VAL] [TC1=VAL] [TC2=VAL] [C=VAL] [CRATIO=VAL]
+ [L=VAL] [W=VAL] [M=VAL] [T[EMP]=VAL] [DTEMP=VAL] [SCALE=VAL]
+ [STATISTICAL=0|1]
```

Parameters

- `xx`

  RC wire name.

- `N1, N2`

  Names of the RC wire nodes.

- `MNAME`

  Model name.

- `[R=]VAL`

  Resistance value in Ω. Default is the value of **RES** as specified in the **.MODEL** statement. Specifying **R=** is optional, the value can be specified without it.

- **TC1**=VAL

  First order temperature coefficient of the resistance in $°C^{-1}$. Default is the value of **TC1R** as specified in the **.MODEL** statement.

- **TC2**=VAL

  Second order temperature coefficient of the resistance in $°C^{-2}$. Default is the value of **TC2R** as specified in the **.MODEL** statement.

- **C**=VAL

  Capacitance connected from node N2 to BULK in Farads. See the "RC Wire Model Syntax" on page 150. Default is the value of **CAP** as specified in the **.MODEL** statement. The **C** value is split between the two pins of the resistor. The model parameter **CRATIO** controls the splitting.

- **CRATIO**=VAL

  Controls the splitting of the **C** value between the two pins of the resistor. **C**×**CRATIO** is the value which will be attached to node N1, while **C**×(1.0 – **CRATIO**) will be attached to node N2. Default is 0.0. This parameter cannot be specified for an instance. It can only be specified on the **.MODEL** statement.

- **L**=VAL

  Resistor length in meters. Default is the value of **L** as specified in the **.MODEL** statement.

- **W**=VAL

  Resistor width in meters. Default is the value of **W** as specified in the **.MODEL** statement.

- **M**=VAL

  Multiplier to simulate parallel resistors. Default is 1.

The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**, **MAXL**, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note

Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **T**[**EMP**]=VAL

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **DTEMP**=VAL

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **SCALE**=VAL

  Element scale factor for resistance and capacitance. Default is 1.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

Note

In order to select the RC Wire model, the **LEVEL** parameter must be set to **LEVEL**=3 in the model MNAME.
The model name MNAME is mandatory for the RC wire model. At least one parameter must be specified in this model. If this is not the case, Eldo interprets the definition as a normal resistor.
The same model name MNAME cannot be used for both a normal resistor model and an RC Wire model.

Example

```
r1 n3 n4 mod1 tc1=0.001
```

Specifies an RC wire placed between nodes n3 and n4. The first order temperature coefficient is defined to be 0.001.

RC Wire Model Syntax

```
.MODEL MNAME R[ES] [PAR=VAL]
```

The RC wire model can be made to behave like a simple resistor or an elementary transmission line. This is achieved by specifying an optional capacitor from node `n2` to BULK or ground. The bulk node functions as a ground plane for the wire capacitance. The wire is described by a drawn length and width. The resistance of the wire is the effective length multiplied by the **RSH** parameter then divided by the effective width.

Table 4-7. RC Wire Model Parameters

Nr.	Name	Description	Default	Units
1	LEVEL	Must be set to 3 to select RC wire model		
2	RDEF	Value of resistor. This value has priority over values computed from L and W	1×10^3	Ω
3	POLY[1]	Keyword to identify the resistor as non-linear polynomial		
4	POLYV	Used to change the formula used for computing the R value	2	
5	REVSP	Used to change the formula used for computing the R value	0	
6	BULK	Default reference node for capacitance	0	
7	CAP	Default capacitance	0	F
8	CAPSW	Sidewall fringing capacitance	0	Fm^{-1}
9	COX	Bottomwall capacitance	0	Fm^{-2}
10	DI	Relative dielectric constant	0	
11	DLR	Difference between drawn length and actual length	0	m
12	DW	Difference between drawn width and actual width	0	m
13	L	Default length of wire	0	m
14	RES	Default resistance	0	Ω
15	RSH	Sheet resistance/square	0	
16	SHRINK	Shrink factor	1	
17	TC1C	1st order temperature coefficient for capacitance	0	$°C^{-1}$
18	TC2C	2nd order temperature coefficient for capacitance	0	$°C^{-2}$
19	TC1R	1st order temperature coefficient for resistance	0	$°C^{-1}$
20	TC2R	2nd order temperature coefficient for resistance	0	$°C^{-2}$
21	THICK	Dielectric thickness	0	m

Table 4-7. RC Wire Model Parameters

Nr.	Name	Description	Default	Units
22	W	Default width of wire	0	m
23	KF	Flicker noise coefficient	0.0	
24	AF	Flicker noise exponents	1.0	
25	WEEXP		0.0	
26	LEEXP		0.0	
27	FEXP		1.0	

1. Instead of using the **POLY** keyword, you can also specify parameters `VC1=value` and `VC2=value`. `VC1` is equivalent to the first parameter of the polynomial array, `VC2` is equivalent to the second parameter of the polynomial array.

Calculation of Wire Resistance

The element width and length are scaled by the parameters **SCALE** and **SHRINK**. The model width and length are scaled by the parameters **SCALM**=x (**.OPTION** command) and **SHRINK**. The effective width and length are calculated from the following equations:

$$W_{eff} = W_{scaled} - (2 \times DW_{eff})$$

$$L_{eff} = L_{scaled} - (2 \times DLR_{eff})$$

where:

$$DW_{eff} = DW \times SCALM$$

$$DLR_{eff} = DLR \times SCALM$$

$$L_{scaled} = L \times SHRINK \times SCALM$$

$$W_{scaled} = W \times SHRINK \times SCALM$$

If element resistance R is specified:

$$R_{eff} = \frac{R \times SCALE\ (\text{element})}{M}$$

Otherwise, if $(W_{eff} \times L_{eff} \times RSH)$ is greater than zero, then:

$$Reff = \frac{Leff \times RSH \times SCALE\ (\text{element})}{M \times Weff}$$

If $(W_{eff} \times L_{eff} \times RSH)$ is zero, then:

$$Reff = \frac{RES \times SCALE\ (\text{element})}{M}$$

If AC resistance is specified in the element, then:

$$RACeff = \frac{AC \times SCALE\ (\text{element})}{M}$$

Otherwise, if RAC is specified in the model, RAC is used:

$$RACeff = \frac{RAC \times SCALE\ (\text{element})}{M}$$

If neither are specified, it defaults to:

$$RACeff = Reff$$

If the resistance is less than resmin, it is reset to resmin and a warning message is issued.

$$resmin = \frac{1}{GMAX \times 1000 \times M}$$

Note

`REVSP` and **`POLYV`** usage:
If parameter **`REVSP`** is set to 1 on the **`.MODEL`** card, and if **`POLYV`** is not 2, then the formula for computing the current generated by the Resistor is:
I = V/<rinst>*(1 + P1/2 + P2/3*V*V...)
which leads to a Resistor of value:
R(V) = 1/(dI/dV) = <rinst>/(1 + P1.V + P2*V*V)

Calculation of Wire Capacitance

The effective length is the scaled drawn length less $2 \times DW_{eff}$. L_{eff} represents the effective length of the resistor from physical edge to physical edge. DW_{eff} is the distance from the drawn edge of the resistor to the physical edge of the resistor. The effective width is the same as the width used in the resistor calculation.

$$L_{eff} = L_{scaled} - (2 \times DW_{eff})$$
$$W_{eff} = W_{scaled} - (2 \times DW_{eff})$$

If the element capacitance C is specified:

$$CAP_{eff} = C \times SCALE \times M$$

Otherwise, the capacitance is selected from L_{eff}, W_{eff} and COX.

$$CAP_{eff} = M \times SCALE \times [L_{eff} \times W_{eff} \times COX + 2.0 \times (W_{eff} + L_{eff}) \times CAPSW]$$

If COX is not specified, but the model parameter THICK is not zero, then:

$$COX = \frac{DI \times \varepsilon o}{THICK}$$

if DI is not zero or:

$$COX = \frac{\varepsilon ox}{THICK}$$

if DI = 0, where:

$$\varepsilon o = 8.8542149 \times 10^{-12} \frac{F}{meter}$$

$$\varepsilon ox = 3.453148 \times 10^{-11} \frac{F}{meter}$$

If only model capacitance CAP is specified:

$$CAP_{eff} = CAP \times SCALE \times M$$

Example

```
.model rmodel r cap=0.5p tc1c=0.01 tc2c=0.05
...
r2 n1 n2 r rmodel tc1=0.05
```

Specifies an RC wire placed between nodes n1 and n2 of model type rmodel. Electrical parameters of the model are defined via the **.MODEL** command.

Diffusion Resistor

```
Pxx N1 N2 [NS] MNAME [L=VAL] [W=VAL] [NB=VAL]
```

Parameters

- `xx`

  Diffusion resistor name.

- `N1, N2, NS`

  Names of the diffusion resistor nodes.

- `MNAME`

  Model name; name of the associated **.MODEL** statement.

- `L=VAL`

  Specifies the resistor length in meters. Default is 1µm.

- `W=VAL`

  Specifies the resistor width in meters.

- `NB=VAL`

  Specifies the number of bends in the resistor.

Note

The length L must be explicitly specified either on the instance or in the **.MODEL** command.

Diffusion Resistor Model Syntax

```
.MODEL MNAME R[ES] LEVEL=6 [PAR=VAL]
```

The diffusion resistor model accurately models the effects temperature, applied bias and back-bias dependencies of NWell, N+, and P+ resistors.

Table 4-8. Diffusion Resistor Model Parameters

Nr.	Name	Description	Default	Units
1	L	Drawn length of resistor	-[1]	m
2	W	Drawn width of resistor	1×10^{-6}	m
3	NB	Number of bends in the resistor	0	-
4	M	Multiplicity factor	1.0	

Table 4-8. Diffusion Resistor Model Parameters

Nr.	Name	Description	Default	Units
5	TR,TREF, TNOM	Temperature at which the parameters have been determined	25	°C
6	DTA	Temperature offset of the RDIFF	0	°K
7	TRISE	Alias of DTA resistor element with respect to T_A	0	°K
8	RSHR	Sheet resistance at $T = T_R$	1k	Ω/sq
9	WTOL	Offset between the drawn and effective resistor width	0	m
10	TCR1	Linear temperature coefficient of the resistor	0	$°K^{-1}$
11	TCR2	Quadratic temperature coefficient of the resistor	0	$°K^{-2}$
12	VPR	Reference Pinch-off voltage	0	V
13	SWVP	Coefficient of the width dependence of VP	0	Vm^{-1}
14	POWER	Voltage exponent	1.5	-
15	VDR	Diffusion voltage at $T = T_R$	1	V
16	RINT	Interface resistance at $T = T_R$	0	Ωm^{-1}
17	TCRINTI	Linear temperature coefficient of the interface resistor	0	$°K^{-1}$

1. The length L must be explicitly specified either on the instance or in the `.MODEL` command.

Further information, and equations of the Diffusion resistor model, can be found on the Philips website:
http://www.nxp.com/acrobat_download/other/philipsmodels/resistor.pdf

Semiconductor Resistor

```
Pxx N1 N2 NS MNAME [R=VAL] [L=VAL] [CL=VAL] [W=VAL] [CW=VAL] [AREA=VAL]
+ [STATISTICAL=0|1]
```

Parameters

- xx

  Semiconductor resistor name.

- N1, N2, NS

  Names of the semiconductor resistor nodes.

- MNAME

  Model name.

- **R**=VAL

  Specifies the resistance. (If **R**=0, or if **R** is not specified, then automatic calculations are used.)

- **L**=VAL

  Specifies the resistor length. Default is 10µm.

- **CL**=VAL

  Specifies the contact offset length. Default is 2µm.

- **W**=VAL

  Specifies the resistor width. Default is 2µm (**DEFW** from **.MODEL** statement).

- **CW**=VAL

  Specifies the contact offset width. Default is 0µm.

- **AREA**=VAL

  Specifies the area of the resistor.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

Semiconductor Resistor Model Syntax

```
.MODEL MNAME TYPE [PNAME=PVAL]
```

- **TYPE**

  This must be either **RN** or **RP**.

The figure below illustrates the cross-sectional view of a diffused semiconductor resistor.

Figure 4-2. Cross-Sectional View of Diffused Resistor

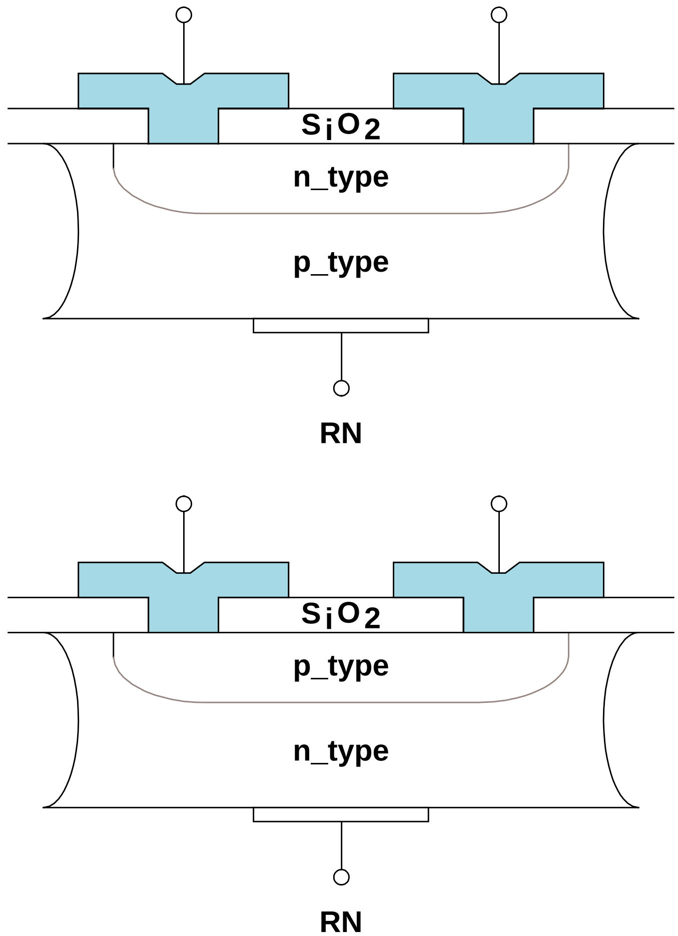


Table 4-9. Semiconductor Resistor Model Parameters

Nr.	Name	Description	Default	Units
1	TC1	First temperature coefficient of bulk	0.0	$\Omega°C^{-1}$
2	TC2	Second temperature coefficient of bulk	0.0	$\Omega°C^{-1}$
3	CTC1	First temperature coefficient of contact hole	0.0	$\Omega°C^{-1}$
4	CTC2	Second temperature coefficient of contact hole	0.0	$\Omega°C^{-1}$
5	RSH	Sheet resistance of bulk	10	Ω/sq
6	CRSH	Sheet resistance of contact hole	10	Ω/sq
7	A1	Contact coefficient 1	1.0	
8	B1	Contact coefficient 2	1.0	
9	DEFW	Default width	2×10^{-6}	m

Table 4-9. Semiconductor Resistor Model Parameters

Nr.	Name	Description	Default	Units
10	NARROW	Narrowing due to side etching	0.0	m
11	IS	Saturation current	1×10^{-14}	A
12	RS	Ohmic resistance	0.0	Ω
13	M	Grading coefficient	0.5	
14	N	Emission coefficient	1.0	
15	TT	Transition time	0.0	s
16	CJO	Zero bias junction capacitance	0.0	F
17	PB	Junction built-in potential	1.0	V
18	EG	Energy gap	1.1	eV
19	XTI (PT)	Saturation current temperature exponent	3.0	
20	KF	Flicker noise coefficient	0.0	
21	AF	Flicker noise exponent	1.0	
22	FC	Forward bias non-ideal junction capacitance coefficient	0.5	
23	BV	Reverse breakdown voltage	•	V
24	IBV	Current at BV	1×10^{-03}	A

Semiconductor Resistor Model Equations

Resistance R calculation

$$R = RBLK + 2RC$$

$$R = RB0 + RB1 + 2RC = R(TNOM)$$

Contact Resistance RC

$$RC = CRSH \cdot \frac{A1}{W^{B1}} = RC(TNOM)$$

Bulk Resistance

$$RBLK = RB0 + RB1$$

- Resistance for bulk part

$$RB0 = \frac{L - 2CL - NARROW}{W - NARROW + 2CW} \cdot RSH = RB0(TNOM)$$

- Resistance near the contact

$$RB1 = \frac{2CL - NARROW}{W - NARROW + 2CW} \cdot CRSH = RB1(TNOM)$$

Temperature Effects

$$R(TEMP) = RB0(TEMP) + RB1(TEMP) + 2RC$$

$$RB0(TEMP) = RB0(TNOM)$$
$$\cdot [1 + TC1 \cdot (TEMP - TNOM) + TC2 \cdot (TEMP - TNOM)^2]$$

$$RB1(TEMP) = RB1(TNOM)$$
$$\cdot [1 + CTC1 \cdot (TEMP - TNOM) + CTC2 \cdot (TEMP - TNOM)^2]$$

- Noise Model Sources

$$i_{nRs} = \left[\frac{4kT}{R}\right]^{\frac{1}{2}}$$

Diode Model Equations

$$T = TEMP + 273.15$$
$$T_0 = TNOM + 273.15$$

Internal DC Model Parameters

$$RATIO = \frac{T - T_0}{T_0}$$

$$q = 1.6\times10^{-19} \quad k = 1.38\times10^{-23} J° K^{-1}$$

$$V_t = \frac{kT}{q} \qquad V_{t0} = \frac{kT_0}{q}$$

$$I_s = IS \cdot AREA \cdot \left[\frac{T}{T_0}\right]^{PT} \left[e^{\left[\frac{(EG)(RATIO)}{V_t}\right]}\right]$$

$$I_{bv} = IBV \cdot AREA$$

$$V_{CRIT} = V_{t0} \ln\left[\frac{V_{t0}}{1.414 \cdot IS}\right]$$

V_{CRIT} is used as a starting point for **DCOP**.

Internal Charge Related Model Parameters

$$Eg = 1.16 - 0.000702\left[\frac{T^2}{T + 1108}\right]$$

$$Eg_{ref} = 1.1150877$$

$$V_{ref} = Vt(T) \cdot \left(\frac{EG(TNOM)}{Vt(TNOM)} - \frac{EG(T)}{Vt(T)} + 3\ln\frac{T}{TNOM}\right)$$

$$P_b = PB\left[\frac{T}{T_0}\right] - V_{ref}$$

$$C_j = CJO \cdot AREA \cdot \left[1 + M\left\{0.0004(T - T_0) + \left[1 - \left(\frac{P_b}{PB}\right)\right]\right\}\right]$$

DC Current Source Definitions

V_d = (Internal Anode Voltage) – (Internal Cathode Voltage)

When BV is not specified by the user:

- If $V_d \geq -5N \cdot V_t$

$$I_d = I_s\left[e^{\left[\frac{V_d}{(N)(V_t)}\right]} - 1\right] + (GMIN)V_d$$

- If $V_d < -5N \cdot V_t$

$$I_d = I_s(e^{-5} - 1) + \left[\frac{I_s}{(N)(V_t)}\right]e^{-5}[V_d + 5N \cdot V_t] + (GMIN)V_d$$

When BV is specified by the user:

- If $V_d \geq -5N \cdot V_t$

$$I_d = I_s\left[e^{\left[\frac{V_d}{(N)(V_t)}\right]} - 1\right] + (GMIN)V_d$$

X_{bv} is an internally generated voltage which facilitates the continuous representation of diode current.

- If $X_{bv} \le V_d < -5N \cdot V_t$

$$I_d = I_s(e^{-5} - 1) + \left[\frac{I_s}{N \cdot V_t}\right]e^{-5}[V_d + 5N \cdot V_t] + (GMIN)V_d$$

- If $V_d < X_{bv}$

$$I_d = I_s(e^{-5} - 1) + \left[\frac{I_s}{N \cdot V_t}\right]e^{-5}[V_d + 5N \cdot V_t]$$

$$- e^{-5}(I_s)\left[e^{\left[\frac{-(V_d - X_{bv})}{N \cdot V_t}\right]} - 1\right] + (GMIN)V_d$$

Diode Diffusion Capacitance

$$C_{d2} = TT\left[\frac{d(I_d)}{d(V_d)}\right]$$

Diode Depletion Capacitance

- If $V_d < FC \cdot P_b$

$$CDS = CS0 \cdot \left(1 - \frac{V_d}{P_b}\right)^{-M}$$

where $CS0 = \frac{1}{2} \cdot \frac{R - 2RC}{RSH} \cdot W^2 \cdot CJ0$

- If $V_d \ge FC \cdot P_b$

$$C_{d1} = C_j\left[(1 - FC)^{-M}\left[\frac{M(1 - FC)^{-M}}{P_b(1 - FC)}\right](V_d - FC \cdot P_b)\right]$$

Small Signal Model Equations

$$C_d = C_{d1} + C_{d2}$$

$$g_d = \frac{d(I_d)}{d(V_d)}$$

$$r_d = \frac{1}{g_d}$$

Diode Model Noise Source Equations

- Resistance Generated Noise

$$i_{nrs} = \left[\frac{4kT}{R_s}\right]^{\frac{1}{2}}$$

- Current Source Generated Shot and Flicker Noise

$$i_{nF} = \left[\frac{KF \cdot I_d^{AF}}{R_s}\right]^{\frac{1}{2}}$$

$$i_{ns} = (2qI_d)^{\frac{1}{2}}$$

Transmission Line

```
Txx NAP NAN NBP NBN [Z0=VAL1] TD=VAL2 [STATISTICAL=0|1]
Txx NAP NAN NBP NBN [Z0=VAL1] F=VAL3 [NL=VAL4] [STATISTICAL=0|1]
```

For Lossy dispersive transmission lines, please refer to "Lossy Transmission Line" on page 165.

Parameters

- `xx`

  Transmission line name.

- `NAP`

  Positive A terminal of the transmission line.

- `NAN`

  Negative A terminal of the transmission line.

- `NBP`

  Positive B terminal of the transmission line.

- `NBN`

  Negative B terminal of the transmission line.

- **TD**=VAL2

  Transmission delay in seconds.

- **F**=VAL3

  Frequency in Hertz.

- **Z0**=VAL1

  Characteristic impedance in Ω. Default value is 50Ω. Optional.

- **NL**=VAL4

  Number of wavelengths at frequency **F** required for a wave to propagate down the line. Default value is 0.25. Optional.

  The above parameters can also be assigned via the **.PARAM** command.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

The relationship between the **TD**, **F** and **NL** values is given by:

$$TD = \frac{1}{F} \times NL$$

Examples

```
t1 1 2 3 4 zo=220 td=111ns
```

Specifies a transmission line t1 between nodes 1 (+ve A) and 2 (-ve A) and nodes 3 (+ve B) and 4 (-ve B), with a characteristic impedance of 220 Ω. The transmission delay of the line is 111 ns.

```
t2 1 2 3 4 zo=220 f=2.25meg
```

Specifies the same transmission line as above, but in terms of a frequency for a quarter wavelength (as **NL** defaults to 0.25).

```
t3 1 2 3 4 zo=220 f=4.5meg nl=0.5
```

Specifies the same transmission line as above, but in terms of a frequency for half a wavelength.

Lossy Transmission Line

LDTL (Lossy Dispersive Transmission Line) is a model implemented in Eldo to simulate transmission lines. It is dedicated for the simulation of lossy coupled uniform lines, also including dispersive effects. This model can be used in all analysis modes (DC, AC, Transient, SST, SSTNOISE, or MODSST). To specify the line parameters to the LDTL model, four different inputs are available:

1. the first level corresponds to R, L, C and G matrices,
2. the second level uses a file at XFX output format as input to specify the line parameters,
3. the third level corresponds to the electrical parameters for a single line,
4. the fourth level corresponds to the geometrical and physical parameters for a single stripline or up to two coupled microstrip lines.

Ways of instantiation are shown on the following pages. You can also instantiate an LDTL model by using a model of MODFAS type (see “.MODEL” on page 723) directly in the parameter list. This model must be specified at the end of the instantiation line and can contain any parameter. For each level, an example is provided in the directory *$MGC_AMS_HOME/examples/tlines/LDTL_model*.

Level 1

```
Yxx LDTL [PIN:] P1...PN [REFin] PN+1...P2N REFout
+ [PARAM: [LEVEL=1] [LENGTH=val] [SAVEFIT=val] [M=VAL]
+ [R(i)=val] [L(i,j)=val] [C(i,j)=val] [G(i,j)=val] [FR1=val]]
```

The first level is dedicated to the simulation of an infinite number of coupled transmission lines. The Maxwell matrices (R, L, C and G) are used to describe the line system.

Parameters

- `xx`

  Transmission line name.

- `P1...PN`

  The N nodes at one end of the line system for a system consisting of N lines.

- `REFin`

  Optional reference node for input signal, used to simulate differential lines.

- `PN+1...P2N`

  The N nodes at the other end of the line system. The line number i in the line system connects the nodes Pi and PN+i.

- `REFout`

  Reference node for output signal. If `REFin` is not specified, then both sides of the line system have the same reference plane.

- **`LENGTH`**`=val`

  Geometric length of the line system. Default value is 1 m. If **`LENGTH`**=0, Eldo uses the default value.

- **`LEVEL`**`=1`

  Keyword to specify the input format. Value 1 specifies the "R,L,G,C matrices" format. Default value is 1.

- **`SAVEFIT`**`=val`

  `SAVEFIT`=1 ⇒ Saves the initialization of the transmission line model (in the file *circuit_name.fit*), in order to speed up the following simulations of the same netlist. Default value is 0.

- **`M`**`=val`

  Device multiplier. Simulates the effect of multiple devices in parallel. In effect the current value is multiplied by **`M`**. Default is 1. **`.OPTION YMFACT`** must be selected in order for this option to work.

- **`R(`**`i`**`)`**`=val`

  Value of the (i,i) element per unit length of the resistance matrix: R. Default values are 50 Ωm^{-1}.

- **`L(`**`i,j`**`)`**`=val`

  Value of the (i,j) element per unit length of the inductance matrix: L. Default values $1\times10^{-6}Hm^{-1}$ for the self inductance and 0 for the mutual inductances.

- **`C(`**`i,j`**`)`**`=val`

  Value of the (i,j) element per unit length of the capacitance matrix: C. Default values $1\times10^{-9}Fm^{-1}$ for the self capacitance and 0 for the mutual capacitances.

- **`G(`**`i,j`**`)`**`=val`

  Value of the (i,j) element per unit length of the conductance matrix: G. Default values are 0 Sm^{-1}.

- `FR1=val`

  Frequency at which dispersion starts (only affects resistance). Default: no dispersion will be considered.

If for a line only `R(1)` is specified, this value is used for all lines. Specification of the Transmission line matrix parameter can be done in one of the following ways:

- Complete matrix of coefficients consisting of N×N values.
- Only the upper (or lower) triangular matrix because of the matrix symmetry.

- Only the first row (or column) of the matrix. This is normally sufficient if all lines in the system have the same width and spacing.

Example

Circuit name: *YLDTL_level1_example.cir*. Three coupled lines defined by R, L, C and G matrices:

```
Y1 LDTL 2 3 4 0 5 6 7 0
+ param: LEVEL=1 length=0.677
+ R(1)=15 L(1,1)=418n C(1,1)=94p G(1,1)=0.02p
+ R(2)=15 L(2,2)=418n C(2,2)=94p G(2,2)=0.02p
+ C(1,2)=-22p C(2,3)=-22p
+ L(1,2)=125n L(2,3)=125n
+ R(3)=15 L(3,3)=418n C(3,3)=94p G(3,3)=0.02p
```

Notice that C(1,2) and C(2,3) are both negative. This is because Eldo uses the Maxwell matrix. The capacitance matrices are based on the admittance matrix of the capacitances between the conductors. The negative values in the capacitance matrix are due to the sign convention for admittance matrices.

Same example using a **`.MODEL`** in the instantiation:

```
Y1 LDTL 2 3 4 0 5 6 7 0
+ param: LEVEL=1 length=0.677
+ model:level1_mod

.model level1_mod MODFAS R(1)=15
+ L(1,1)=418n C(1,1)=94p G(1,1)=0.02p
+ R(2)=15 L(2,2)=418n C(2,2)=94p G(2,2)=0.02p
+ C(1,2)=-22p C(2,3)=-22p
+ L(1,2)=125n L(2,3)=125n
+ R(3)=15 L(3,3)=418n C(3,3)=94p G(3,3)=0.02p
```

Level 2

```
Yxx LDTL [PIN:] P1...PN [REFin] PN+1...P2N REFout
+ [PARAM: [LEVEL=2] [LENGTH=val] [SAVEFIT=val]
+ [XFX_IDF=val] [FP=val] [MULTIDEBYE=val]]
```

The second level uses a file of XFX output format as input to specify the line parameters. This level is dedicated to the simulation of an infinite number of coupled transmission lines.

Parameters

The description of the global parameters (`PIN`s, **`LENGTH`** and **`SAVEFIT`**) is as specified for the **`LEVEL`**=1 format. Level 2 specific parameters are shown below.

- **`LEVEL`**`=2`

  Keyword to specify the input format. Value 2 specifies the XFX format. Default value is 1.

- **XFX_IDF**=val

  Index relative to the XFX file name. If XFX_IDF=val, Eldo will search for the file *XFX_val.tlp*. Eldo searches for the specified file in a specific order, see "Search path priorities" on page 694.

- **FP**=val

  Polarization frequency to control dispersive effect on the conductance (see "Technical Precision" on page 173). Default: 1.6×10^9.

- **MULTIDEBYE**=val

  val=1 specifies the use of multi-pole debye model to model the dispersive effect on the conductance (recommended for modeling PCB-type dielectrics). val=0, this model is not used. (see "Technical Precision" on page 173). Default: 1.

Example

Circuit name: *YLDTL_level2_example.cir*. Two coupled lines defined by an XFX output file:

```
Y1 LDTL 1 2 3 4 0
+ param: LEVEL=2 length=0.1  xfx_idf=12
```

Same example using a **.MODEL** in the instantiation:

```
Y1 LDTL 1 2 3 4 0
+ param: model:level2_mod
.model level2_mod MODFAS LEVEL=2 length=0.1  xfx_idf=12
```

The parameter xfx_idx=12 is a reference to the file *XFX_12.tlp* containing the line information generated by XFX. Here is an example of such a file:

```
XFX V6.0.0.0 Report         5 Nov 15:57 1997      Setup File=ex3.xfx

Configuration Name: ANALOG1  Conductors: 2

Conductor index: 0     name: $$GND$$
Conductor index: 1     name: A
Conductor index: 2     name: C

 i    j     Lij      Cij      Ze     Zo      Se      So     Fwdx  Rvsx
from to  (nh/in) (pf/in)  (ohms) (ohms)  (ns/ft) (ns/ft) (s/s) (v/v)
---------------------------------------------------------------------
 1   1   8.631     3.742     48.02    -     2.16    -        -     -
 1   2   6.87e-10  2.98e-10  48.02  48.02   2.16   2.16   0.000 0.000
 2   2   8.631     3.742     48.02    -     2.16    -        -     -

:     LOSS MATRICES
 i    j        Rsij          Gij      Rdcij     Gdcij
from  to   (ohm-nsec^.5)  (mS-ns)    (ohms)    (mS)          PER INCH
--------------------------------------------------------------------
  1   1     0.7             0.0      0.47929   0.46827
  1   2     0.00            0.0      0.00000   0.02912
  2   2     0.7             0.0      0.34473   0.45333
;
```

Level 3

```
Yxx LDTL [PIN:] P1 [REFin] P2 REFout
+ [PARAM: [LEVEL=3] LENGTH=val] [SAVEFIT=val]
+ [Zc=val] [VREL=val] [TD=val] [L=val] [C=val] [R=val] [FR1=val] [M=val]]
```

The third level corresponds to the electrical parameters for a single line only.

Parameters

The description of the global parameters (`PIN`s, **`LENGTH`** and **`SAVEFIT`**) is as specified for the **`LEVEL`**=1 format. Level 3 specific parameters are shown below.

- **`LEVEL`**`=3`

  Keyword to specify the input format. Value 3 specifies the electrical format. Default value is 1.

- **`Zc`**`=val`

  Characteristic impedance (Ω). If this value is not specified, it is calculated with the values of L and C.

- **`VREL`**`=val`

  Relative velocity. If this value is not specified, it is calculated with the values of L and C.

- **`TD`**`=val`

  Delay for **`LENGTH`** (implies total delay calculated is **`LENGTH`**×**`TD`**). If not specified, calculated.

- **`L`**`=val`

  Inductance per unit length. Default value: $1\times10^{-6}Hm^{-1}$.

- **`C`**`=val`

  Capacitance per unit length. Default value is: $1\times10^{-9}Fm^{-1}$.

- **`R`**`=val`

  Linear resistance. Default value is 50 Ωm^{-1}.

- **`FR1`**`=val`

  Frequency (Hz) at which dispersion starts (only affects resistance). Default: no dispersion will be considered.

- **`M`**`=val`

  Device multiplier. Simulates the effect of multiple devices in parallel. In effect the current value is multiplied by **`M`**. Default is 1. **`.OPTION YMFACT`** must be selected in order for this option to work.

You can either specify directly the line parameters **`R`**, **`L`** and **`C`**, or use any combination of electrical parameters (L and C can be computed with electrical parameters, see “Technical Precision” on page 173).

Example

Circuit name: *YLDTL_level3_example.cir*. One dispersive line defined by electrical parameters:

```
Y1 ldtl 1 2 0
+ param: LEVEL=3 length=100 R=1
+ ZC=50 VREL=0.66 FR1=100Meg
```

Same example using a **.MODEL** in the instantiation:

```
Y1 ldtl 1 2 0
+ param: LEVEL=3 length=100 R=1
+ ZC=50 model: level3_mod

.model leve3_mod MODFAS VREL=0.66 FR1=100Meg
```

Level 4

```
Yxx LDTL [PIN:] P1 [REFin] P2 REFout
+ [PARAM: [LEVEL=4] [LENGTH=val [SAVEFIT=val]
+ [DLEV=val] [PLEV=val] [ER=val] [H=val] [W=val] [T=val]
+ [RHO=val] [TAND=val] [H1=val] [FP=val] [H2=val] [S=val]
+ [THICKNESS=val] [DISPERSIVE=val] [USE_ER=val] [M=val] [MULTIDEBYE=val]]
```

The fourth level corresponds to the geometrical parameters specification for microstrip line and stripline. Up to two coupled microstrip lines are allowed and only one single stripline can be taken into account. The structure of these transmission lines is illustrated in Figure 4-4 (coupled pair of microstrip lines) and Figure 4-5 (stripline).

Parameters

The description of the global parameters (PINs, **LENGTH** and **SAVEFIT**) is as specified for the **LEVEL**=1 format. Level 4 specific parameters are shown below.

- **LEVEL**=4

  Keyword to specify the input format. Value 4 specifies the geometrical and physical format. Default value is 1.

- **DLEV**=val

  Type of line: 1 for microstripline; 2 for stripline. Default is 1.

- **PLEV**=val

  Type of equations: 0 uses the equations from the references (1) and (2), 1 for simplified equations. Default is 0.

- **ER**=val

  Dielectric relative permittivity. Default value is 9.8 (alumina).

- **H**=val

  Dielectric thickness (m). Default value is 400×10^{-6}m.

- **W**=val

  Conductor width. Default value is 50×10^{-6} m.

- **T**=val

  Conductor thickness. Default value is 5×10^{-6} m.

- **RHO**=val

  Conductor resistivity. Default value is 17×10^{-9} Ωm (copper).

- **TAND**=val

  Dielectric loss tangent. Default value is 0.

- **H1**=val

  Conductor height, only for stripline configuration. Default value is 197.5×10^{-6} m.

- **FP**=val

  Polarization frequency to control dispersive effect on the conductance (see "Technical Precision" on page 173). Default: 1.6×10^{9}.

- **H2**=val

  Height between dielectric and a possible cover plate. Only for coupled microstrip configuration. Default: 0.0, means that no cover plate is taken into account.

- **S**=val

  Spacing between the two conductors. Default = conductor width value (W). Only for coupled microstrip configuration.

- **THICKNESS**=val

  Take into account effect of finite strip thickness if `val`=1. Default: 0. Only for coupled microstrip configuration.

- **DISPERSIVE**=val

  Take into account dispersive effect if `val`=1. Default: 0. Only for coupled microstrip configuration.

- **USE_ER**=val

  Use directly the dielectric relative permittivity (ER) to compute the characteristic impedance (Zc) if `val`=1. Otherwise (if `val`= 0), an effective relative permittivity will be calculated and used in the Zc computation (see Technical Precision, "Level 4" on page 175 for details). Default: 1.

- **M**=val

  Device multiplier. Simulates the effect of multiple devices in parallel. In effect the current value is multiplied by **M**. Default is 1. **.OPTION YMFACT** must be selected in order for this option to work. Only used for stripline (**DLEV**=2).

- **MULTIDEBYE**=val

  val=1 specifies the use of multi-pole debye model to model the dispersive effect on the conductance (recommended for modeling PCB-type dielectrics). val=0, this model is not used. (see “Technical Precision” on page 173). Default: 1.

Note

When **PLEV**=0 and **DLEV**=2, it is only possible to describe an off-centered microstrip line with the equations based on reference [1].

Examples

Circuit name: *YLDTL_level4_example.cir*. A microstrip line based on equations from reference [1].

```
Y1 LDTL 1 2 0
+ param: LEVEL=4 length=10 PLEV=0 DLEV=1
+ h=400u w=50u t=5u rho=17E-09 er=9.8
```

Circuit name: *YLDTL_level4_example2.cir*. Symmetric pair of coupled microstrip lines, including finite strip thickness and dispersive effects.

```
Y1 LDTL 1 2 0 3 4 0
+ param: LEVEL=4 length=10e-3 PLEV=1 DLEV=1
+ h=635u w=88u t=2u s=90u h2=935u
+ rho=1.72E-08 tand=0.01 thickness=1
+ dispersive=1
```

Same example using a **.MODEL** in the instantiation:

```
Y1 LDTL 1 2 0 3 4 0
+ param: LEVEL=4 length=10 PLEV=1 DLEV=1
+ h=635u w=88u model:level4_mod

.model level4_mod MODFAS t=2u s=90u h2=935u
+ rho=1.72E-08 tand=0.01 thickness=1
+ dispersive=1
```

Error Message Treatment

A general problem which causes many errors is incorrect time delay values. These values are calculated by multiplying the L and C matrices. Negative or null time delay values can cause errors in the model. To avoid the simulation being stopped, here follows some advice on how to avoid errors. The error message is shown, together with advice on avoiding this type of error.

```
ERROR model Yxx : no time-delay in the transmission line(s), check your
input parameters.
```

This means that some value(s) of the L or C matrix (or both) are bad: the diagonal of L×C matrix presents null value(s) (see “General Equation for Delay” on page 173). It can appear when some off-diagonal term(s) of the C or L matrix are too large. Normally the coupling effect on L and C decreases when the distance between the two concerned lines increase. That means: |C(1,2)| should be larger than |C(1,3)|.

```
ERROR model Yxx: Non physical line model (negative time delay). Check C
and/or L matrices off-diagonal terms.
```

This message appears only for coupled transmission line models. It means that the time-delay of at least one line of the model is negative. So this modelization is not a physical one.

This means the diagonal of L×C matrix presents negative value(s) (see "General Equation for Delay" on page 173). It can appear when some off-diagonal term(s) of the C or L matrix are too large. Normally the coupling effect on L and C decreases when the distance between the two concerned lines increase. That means: |C(1,2)| should be larger than |C(1,3)|.

```
ERROR model Yxx : the diagonal of C is non-strictly-dominant : you should
have Sum{|(C(i,j)|} < |C(i,i)| (i != j)
```

This means some off-diagonal terms of the C matrix are too large. Therefore, the sum of all the off-diagonal terms of one line of the matrix is not lower than the diagonal term of the line: the strictly-dominant property is not verified. Such a property is required for the model.

```
WARNING model Y1 : negative diagonal value(s) : R[1][1]
```

This warning means that the value R[1][1] is negative. Therefore, you have to check the parameters of the model instantiation according to the LEVEL used (see Technical Precision).

Technical Precision

Here follows some technical information about the use of the `Yxx LDTL` model.

General Equation for Delay

The time-delay (*Td*) of a single transmission line is computed as follows:

$$Td = Length \times \sqrt{LC}$$

When we have a n-coupled transmission line model, the time-delay matrix is computed as follows (*L*, *C* and *Tdm* are matrices):

$$Tdm = Length \times diag(L \times C)$$

Tdm is a diagonal matrix. The n^{th} element of the diagonal is the time-delay of the n^{th} line of the model. Therefore, these diagonal values must be positive.

Level 1

To introduce skin effects in the line model (loss that is proportional to the square root of frequency), you just have to specify the *FR1* parameter. Then the resistivity value will be frequency dependent:

$$R = R(i) \times \left(1 + (1+i)\sqrt{\frac{f}{FR1}}\right)$$ for the ***i***th transmission line.

Level 2

Here the skin effect is introduced by the parameter *Rs*:

$$R = R_{DC} + R_S \times (1+i)\sqrt{4\pi f}$$

You can also introduce frequency dependent conductance by using the parameters *Gs* and *fp* (polarization frequency: **FP** parameter). The conductance dispersive effect can be modeled in two ways according to the **MULTIDEBYE** parameter:

- One-pole debye model (**MULTIDEBYE**=0)

  The dispersive effect is obtain according to the following equation:

  $$G = G_{DC} + G_S \times \frac{i2\pi f \times 2\pi fp}{i2\pi f + 2\pi fp}$$

- Multi-pole debye model (**MULTIDEBYE**=1)

  By using this option, we build a complex frequency-dependent capacitance matrix:

  $$C(\omega) = C_{inf} + f(j\omega)C_d$$

  Therefore, line conductance per unit length becomes:

  $$Y(j\omega) = G_{DC} + (j\omega)C_{inf} + (j\omega)f(j\omega)C_d$$

  where: $C_{inf} = C - \alpha G_s$ and $C_d = \beta G_s$; *C* and *Gs* are the user-defined matrices.

  with $$\alpha = \frac{\ln\left(\frac{\omega_2^2 + \omega_0^2}{\omega_1^2 + \omega_0^2}\right)}{4\pi\left(\operatorname{atan}\left(\frac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\frac{\omega_0}{\omega_2}\right)\right)}$$ and $$\beta = \frac{\ln(10^8)}{2\pi\left(\operatorname{atan}\left(\frac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\frac{\omega_0}{\omega_2}\right)\right)}$$

  Finally, the function $$f(j\omega) = \frac{\ln\left(\frac{\omega_2 + j\omega}{\omega_1 + j\omega}\right)}{\ln(10^8)}$$ which is fitted with 15 real poles.

  Note: $\omega_0 = 2\pi f_p$, $\omega_1 = 10^4$, $\omega_2 = 10^{12}$, $\omega = 2\pi f$ and *fp* the polarization frequency (**FP** parameter).

Level 3

As already described, you can either specify directly the line parameters **R**, **L** and **C**, or use any combination of electrical parameters. In order to discard redundant parameter sets we use the following equations:

Table 4-10. LDTL Level 3 Parameter Combinations

Input Parameters	Equations
Zc, VREL	$C = \frac{1.0}{Zc \times VREL \times Clight}, L = \frac{Zc}{VREL \times Clight}$
Zc, TD	$C = \frac{TD}{Zc}, L = Zc \times TD$
TD, C	$L = \frac{TD^2}{C}$
VREL, C	$L = \frac{1.0}{VREL^2 \times Clight^2 \times C}$
TD, L	$C = \frac{TD^2}{L}$
VREL, L	$C = \frac{1.0}{VREL^2 \times Clight^2 \times L}$
any other	default values for *L* and *C*

Clight = 3×10^8 ms^{-1} (the speed of light)

The skin effect is introduced by the parameter **FR1**:

$$R = R \times \left(1 + (1 + i)\sqrt{\frac{f}{FR1}}\right)$$

Level 4

Only two transmission line configurations can be described with this input format: microstrip line and stripline. The following figures provides the structure of these transmission lines. Figure 4-3 and Figure 4-4 provide the structure for a single and covered pair of microstrip models, Figure 4-5 provides the structure for a stripline.

Figure 4-3. Microstrip Line Structure

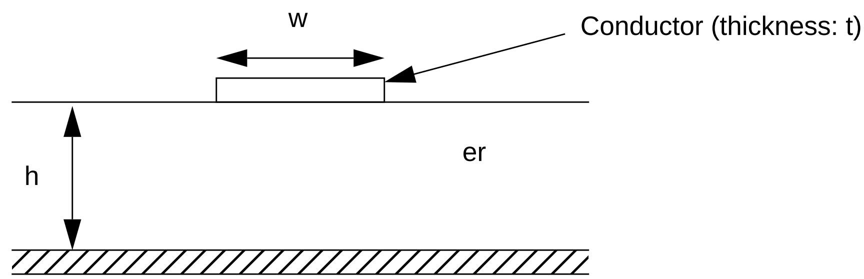


Figure 4-4. Covered Pair Microstrip Line Structure

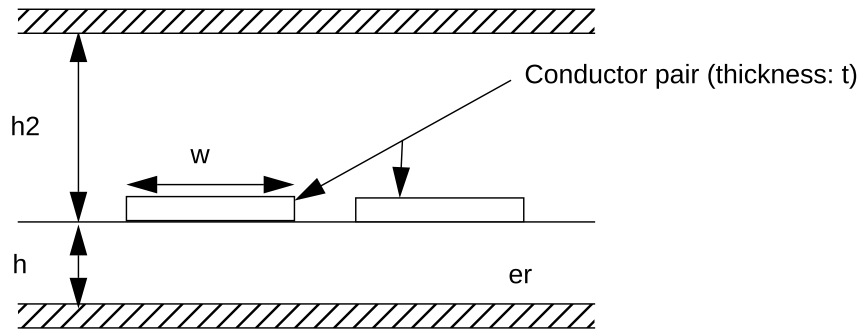
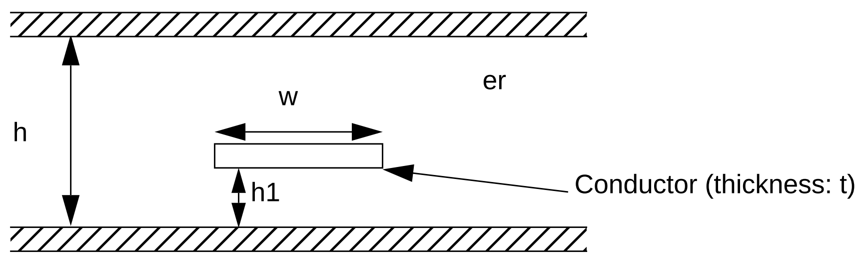


Figure 4-5. Stripline Structure



The **PLEV** parameter allows the use of two sorts of equation: reference equations (**PLEV**=0) and simplified equations (**PLEV**=1). Provided here is the reference formulation.

For a Single Microstrip Line (DLEV=1)

- From reference (1) (**PLEV**=0), we have the following equations:

$$\text{Capacitance } C = \frac{\sqrt{\mu_o \varepsilon_o er}}{Zc}$$

$$\text{Inductance } L = Zc\sqrt{\mu_o \varepsilon_o er}$$

$$\text{Resistance } R = \frac{rho}{w \times t}$$

with the characteristic impedance:

$$Zc = \frac{\eta_o}{2\sqrt{2}\pi\sqrt{er+1}}\ln\left\{1 + \frac{4h}{w'}\left[\frac{14+8/(er)}{11} \times \frac{4h}{w'} + \sqrt{\left(\frac{14+8/er}{11}\right)^2\left(\frac{4h}{w'}\right)^2 + \frac{1+1/er}{2}\pi^2}\right]\right\}$$

where:

$$w' = w + \Delta w'$$

$$\Delta w' = \Delta w\left(\frac{1+1/(er)}{2}\right)$$

$$\Delta w = \frac{1}{\pi}\ln\left[\frac{4e}{\sqrt{(t/h)^2 + \left(\frac{1/\pi}{w/t+1.1}\right)^2}}\right]$$

With η_o, the wave impedance; ε_o, the permittivity; and μ_o the permeability of free space.

In reference (1), it is recommended to replace Er by Eeff in the characteristic impedance computation. It is done by specified the parameter **USE_ER**=0.

for $\frac{w}{h} \leq 1$:

$$E_{eff} = \frac{Er+1}{2} + \frac{Er-1}{2}\left[\left(1+\frac{12h}{w}\right)^{-0.5} + 0.04\left(1-\frac{12h}{w}\right)^2\right]$$

and for $\frac{w}{h} \geq 1$:

$$E_{eff} = \frac{Er+1}{2} + \frac{Er-1}{2}\left(1+\frac{12h}{w}\right)^{-0.5}$$

For a Symmetric Pair of Coupled Microstrip Lines (DLEV=1)

- From reference (2) (**PLEV**=0), we have the following equations:

$$\text{Capacitance matrix} \begin{bmatrix} \hat{C} + \hat{C}_M & -\hat{C}_M \\ -\hat{C}_M & \hat{C} + \hat{C}_M \end{bmatrix}$$

where: $\hat{C} = \frac{1.0}{v_{p,e} \times Z_{L,e}}$ and $\hat{C}_M = \frac{1}{2}\left(\frac{1.0}{v_{p,o} \cdot Z_{L,o}} - \frac{1.0}{v_{p,e} \cdot Z_{L,e}}\right)$

$$\text{Inductance matrix} \begin{bmatrix} \hat{L} & \hat{L_M} \\ \hat{L_M} & \hat{L} \end{bmatrix}$$

$$\text{where: } \hat{L} = \frac{1}{2}\left(\frac{Z_{L,e}}{v_{p,e}} + \frac{Z_{L,o}}{v_{p,o}}\right) \text{ and } \hat{L_M} = \frac{1}{2}\left(\frac{Z_{L,e}}{v_{p,e}} - \frac{Z_{L,o}}{v_{p,o}}\right)$$

$$\text{Resistance matrix} \begin{bmatrix} \frac{Re+Ro}{2} & \frac{Re-Ro}{2} \\ \frac{Re-Ro}{2} & \frac{Re+Ro}{2} \end{bmatrix}$$

$$\text{where: } Re = \frac{\ln 10}{10} \cdot \alpha_{c,e} \cdot Z_{L,e} \text{ and } Ro = \frac{\ln 10}{10} \cdot \alpha_{c,o} \cdot Z_{L,o}$$

$$\text{Conductance matrix} \begin{bmatrix} \frac{Ge+Go}{2} & \frac{Ge-Go}{2} \\ \frac{Ge-Go}{2} & \frac{Ge+Go}{2} \end{bmatrix}$$

$$\text{where: } Ge = \frac{\ln 10}{10} \cdot \frac{\alpha_{c,e}}{Z_{L,e}} \text{ and } Go = \frac{\ln 10}{10} \cdot \frac{\alpha_{c,o}}{Z_{L,o}}$$

The indices *e* and *o* indicate even and odd mode parameters respectively. The expression of effective relative permittivity, characteristic impedance and attenuation coefficients are given in the single microstrip line description (for more details, see reference (2)). Dispersion is taken into account (when required) in the effective permittivity and characteristic impedance computations (see reference (2)). Therefore, effective parameter matrices (R, L, C and G) values change with frequency.

For a Single Stripline (DLEV=2)

- From reference (1) (**PLEV**=0), we have the following equations:

$$\text{Capacitance } C = \frac{\sqrt{\mu_0 \varepsilon_0 er}}{Z_0}$$

$$\text{Inductance } L = Z_0 \sqrt{\mu_0 \varepsilon_0 er}$$

$$\text{Resistance } R_{DC} = \frac{rho}{w \times t}$$

with: $$Z_0 = \frac{\eta_0}{\sqrt{er} \times \frac{C_1}{\varepsilon}}$$

and:

$$\frac{C_1}{\varepsilon} = \frac{2w_1/h}{1 - s/h - t/h} + \frac{2w_1/h}{1 + s/h - t/h} +$$
$$\frac{2}{\pi}\left(\frac{2}{1 - t/(h-s)}\ln\left[1 + \frac{1}{1 - t/(h-s)}\right] + \left(1 - \frac{1}{1 - t/(h-s)}\right)\ln\left[\frac{1}{(1 - t/(h-s))^2} - 1\right]\right) +$$
$$\frac{2}{\pi}\left(\frac{2}{1 - t/(h+s)}\ln\left[1 + \frac{1}{1 - t/(h+s)}\right] + \left(1 - \frac{1}{1 - t/(h+s)}\right)\ln\left[\frac{1}{(1 - t/(h+s))^2} - 1\right]\right)$$

where:

$s = h - (2 \times h1 + t)$, for a centered stripline s = 0

if $\left(\frac{w}{h-t}\right) < 0.35$ then $w_1 = \left(\frac{0.07 \times (h-t) + w}{1.2}\right)$

else $w_1 = w$

Dispersive effects are introduced according to the value of the geometrical parameters. So the resistivity can be frequency dependent:

$$R = R_{DC} \times \left(1 + (1 + i) \times \sqrt{\frac{\frac{f}{rho}}{\pi\mu_0 t^2}}\right)$$

You can also introduce frequency dependent conductance by using the parameters *Gs* and *fp* (polarization frequency: **FP** parameter). The conductance dispersive effect can be modeled in two ways according to the **MULTIDEBYE** parameter:

- One-pole debye model (**MULTIDEBYE**=0)

  $$G = \tan d \times C \times \frac{i2\pi f \times 2\pi fp}{i2\pi f \times 2\pi fp}$$

  where *fp* is the polarization frequency (**FP** parameter).

- Multi-pole debye model (**MULTIDEBYE**=1)

  By using this option, we build a complex frequency-dependent capacitance matrix:

  $$C(\omega) = C_{inf} + f(j\omega)C_d$$

Therefore, line conductance per unit length becomes:

$$Y(j\omega) = (j\omega)C_{inf} + (j\omega)f(j\omega)C_d$$

where: $C_{inf} = C - \alpha \tan dC$ and $C_d = \beta \tan dC$; C is the user-defined matrix and *tand* the dielectric loss tangent parameter.

$$\text{with } \alpha = \frac{\ln\left(\dfrac{\omega_2^2 + \omega_0^2}{\omega_1^2 + \omega_0^2}\right)}{4\pi\left(\operatorname{atan}\left(\dfrac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\dfrac{\omega_0}{\omega_2}\right)\right)} \text{ and } \beta = \frac{\ln(10^8)}{2\pi\left(\operatorname{atan}\left(\dfrac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\dfrac{\omega_0}{\omega_2}\right)\right)}.$$

Finally, the function $f(j\omega) = \dfrac{\ln\left(\dfrac{\omega_2 + j\omega}{\omega_1 + j\omega}\right)}{\ln(10^8)}$ which is fitted with 15 real poles.

Note: $\omega_0 = 2\pi f_p$, $\omega_1 = 10^4$, $\omega_2 = 10^{12}$, $\omega = 2\pi f$ and *fp* the polarization frequency (**FP** parameter).

Reference

(1) Transmission Line Design Handbook, Brian C. Wadell, Artech House 1991.
(2) Implementation of Single and Coupled Microstrip Line in APLAC, Luis Costa and Martti Valtonen, CT-33 December 1997.

Lossy Transmission Line: W Model

The W model is implemented in Eldo to simulate lossy coupled uniform lines including dispersive effects. This model can be used in all analysis modes (DC, AC, Transient, SST, SSTNOISE, or MODSST). The general instantiation of a W model is shown below. Examples are provided in the directory *$MGC_AMS_HOME/examples/tlines/W_model*.

RLGCfile Form

```
Wxx N=nb_line
+ P1...PN PGNDin PN+1...P2N PGNDout
+ RLGCfile=file_name L=length [FP=val]
+ [MULTIDEBYE=val] [SAVEFIT=val] [COMPAT=val] [FGD=val]
```

Umodel Form

```
Wxx N=nb_line
+ P1...PN PGNDin PN+1...P2N PGNDout
+ Umodel=model_name L=length [SAVEFIT=val]
```

RLGCmodel Form

```
Wxx N=nb_line
+ P1...PN PGNDin PN+1...P2N PGNDout
+ RLGCmodel=model_name L=length [FP=val]
+ [MULTIDEBYE=val] [SAVEFIT=val] [COMPAT=val] [FGD=val]
```

Tabular RLGCmodel Form

```
Wxx P1...PN PGNDin PN+1...P2N PGNDout
+ N=nb_line L=length
+ TABLEMODEL=table_model_name [SAVEFIT=val] [FITTABLEMODEL=val]
```

Parameters

- `xx`

  W model transmission line name.

- `N=nb_line`

  Number of lines.

- `P1…PN`

  The N nodes at one end of the line system for a system consisting of N lines.

- `PGNDin`

  Reference node for the P1…PN nodes of the line system.

- `PN+1…P2N`

  The N nodes at the other end of the line system. The line number i in the line system connects the nodes Pi and PN+i.

- `PGNDout`

  Reference node for the PN+1…P2N nodes of the line system.

- `RLGCfile=file_name`

  Name of the file containing R, L, C, G, R_s and G_d matrices.

- `Umodel=model_name`

  Name of the transmission line model. This entry allows the use of the U model (see "Lossy Transmission Line: U Model" on page 192) entries in the W model.

- `TABLEMODEL=table_model_name`

  Name of the model containing R, L, C and G tabular matrices description.

- `L=length`

  Geometric length of the system (meter). Default value is 1.0. If `L`=0, Eldo uses the default value.

- **`FP`**`=val`

  Polarization frequency to control dispersive effect on the conductance. Default: 1.6×10^9.

- `MULTIDEBYE=val`

  `val`=1 specifies the use of multi-pole debye model to model the dispersive effect on the conductance (recommended for modeling PCB-type dielectrics). `val`=0, this model is not used. Default: 1.

- `SAVEFIT=val`

  If the value is 1, this option saves the initialization of the transmission line model (in the file *circuit_name.fit*), in order to speed up the following simulations of the same netlist. Default value is 0.

- `COMPAT=val`

  If the value is 1, it specifies the model used for the dispersive effect is based on conductance, see the formula details of each W model instantiation: "RLGC File Syntax" on page 183 and "RLGC Model Syntax" on page 186. Default value is taken from the global option **`COMPAT`** (if **`.OPTION COMPAT`** is specified then `COMPAT`=1). Note that the `MULTIDEBYE` parameter priority is higher than `COMPAT`. If `MULTIDEBYE` is specified, then the value of `COMPAT` is zero.

- `FGD=val`

  Cut-off frequency value. Default is zero. Can only be specified in compat mode (**`COMPAT`**=1).

- `FITTABLEMODEL=val`

  When set to 1, it will enable a causal model (admittance and propagation) to be built from non-causal tabulated data. If set to 0 (default), the tabulated data will not be modified to build the model and the built-in models are considered to be causals.

Examples

RLGCfile Entry

Circuit name: *RLGCfile_example.cir*.

```
W1 N=2
+ 1 2 0 3 4 0
+ RLGCfile=2lin.rlgc L=0.97e-3
```

See “Example RLGC file” on page 185.

Umodel Entry

```
.MODEL uname1 U LEVEL=3 ELEV=1 PLEV=1 DLEV=2 NL=1
+ HT=1.0e-4 WD=2.0e-4 TH=5.0e-5 RHO=1.785e-8
W1 N=1 1 0 2 0 Umodel=uname L=1.0e-3
```

RLGCmodel Entry

```
W1 N=2 1 2 0 4 5 0 RLGCmodel=model_rlgc L=0.97e-3
```

Tabular RLGCmodel Entry

```
W1 i1 i2 0 o1 o2 0 N=2 L=0.1 TABLEMODEL=ex1
```

RLGC File Syntax

The RLGC file is a text file, which contains the values of R, L, C, G, R_s and G_d matrices per unit length. This file is order-dependent, and the order is the following:

N — Number of lines.

L_o — DC inductance matrix (per unit length).

C_o — DC capacitance matrix (per unit length).

R_o — DC resistance matrix (per unit length).

G_o — DC conductance matrix (per unit length).

R_s — Skin effect resistance matrix (per unit length):

$$R = R_o + (1 + i)\sqrt{f}Rs$$

G_d — Dielectric-loss conductance matrix (per unit length). The frequency dependent conductance uses the parameters *Gs* and *fp* (polarization frequency: **FP** parameter). It can be modeled in two ways according to the **MULTIDEBYE** parameter:

- One-pole debye model (**MULTIDEBYE**=0)

$$G = G_o + \frac{G_d}{2\pi} \times \frac{i2\pi f \times 2\pi fp}{i2\pi f + 2\pi fp}$$

where *fp* is the polarization frequency (**FP** parameter).

- Multi-pole debye model (**MULTIDEBYE**=1)

  By using this option, we build a complex frequency-dependent capacitance matrix:

  $$C(\omega) = C_{inf} + f(j\omega)C_d$$

  Therefore, line conductance per unit length becomes:

  $$Y(j\omega) = G_o + (j\omega)C_{inf} + (j\omega)f(j\omega)C_d$$

  where: $C_{inf} = C - \alpha G_d$ and $C_d = \beta G_d$; *C* and *Gs* are the user defined matrices, with:

  $$\alpha = \frac{\ln\left(\frac{\omega_2^2 + \omega_0^2}{\omega_1^2 + \omega_0^2}\right)}{4\pi\left(\operatorname{atan}\left(\frac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\frac{\omega_0}{\omega_2}\right)\right)} \text{ and } \beta = \frac{\ln(10^8)}{2\pi\left(\operatorname{atan}\left(\frac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\frac{\omega_0}{\omega_2}\right)\right)}$$

  Finally, the function $f(j\omega) = \frac{\ln\left(\frac{\omega_2 + j\omega}{\omega_1 + j\omega}\right)}{\ln(10^8)}$ which is fitted with 15 real poles.

  Note: $\omega_0 = 2\pi f_p$, $\omega_1 = 10^4$, $\omega_2 = 10^{12}$, $\omega = 2\pi f$ and *fp* the polarization frequency (**FP** parameter).

- Compat dispersive model (**COMPAT**=1)

  $$G = G_o + G_d \times \frac{f}{\sqrt{1 + \left(\frac{f}{f_{gd}}\right)^2}}$$

  where f_{gd} is a cut-off frequency; if f_{gd} value is zero then G keeps linear dependency on the frequency. Default is zero.

The R_o, G_o, R_s and G_d matrices are optional (default value is zero). L_o and C_o matrices must be described in the RLGC file. Since these matrices are symmetrical, only the lower-triangular parts are specified in the RLGC file.

The diagonal terms of L_o and C_o matrices must be positive non-zero; the diagonal terms of R_o, R_s, G_o and G_d matrices must be non-negative. Off-diagonal terms of C_o, G_o and G_d are non-positive.

Comments

A comment line can be specified by an asterisk ‘*’ at the beginning of the line. This comments out the entire line.

Separator

The number can be separated by any combination of the characters shown in the table below:

Table 4-11. RLGC Separator Characters

Character
Space
Tab
New line
,
;
(
)
[
]
{
}

Example RLGC file

Filename: *2lin.rlc*.

```
*RLGC matrices for 2 frequency-dependent lines
*N (number of lines)
********************
2
* Lo
*******
0.3481e-6
0.5458e-7  0.3481e-6
* Co
*******
0.1593e-9
-0.2578e-10  0.1651e-9
* Ro
*******
75
0    50
* Go
*******
0.2421e-3
-0.4860e-4  0.2070e-3
* Rs
*******
0.0025
0    0.0014
```

```
* Gd
*******
1.2e-13
-4.1e-14  1.1e-13
```

RLGC Model Syntax

The RLGC model is a model, which contains the values of R, L, C, G, R_s and G_d matrices per unit length. There is no limitation on the number of coupled lines. Since the matrices are symmetric, only the lower-triangular parts of the matrices have to be described in the RLGC model. Inductance and capacitance matrices (C_o and L_o) have to be specified, the other matrices can be optional.

General Instantiation of the Model

```
.MODEL model_name W MODELTYPE=RLGC N=nb_line
+ Lo=Lo_matrix_entries Co=Co_matrix_entries
+ [Ro=Ro_matrix_entries] [Go=Go_matrix_entries]
+ [Rs=Rs_matrix_entries] [Gd=Gd_matrix_entries]
```

Parameters

- `N=nb_line`

  Number of lines.

- `Lo=Lo_matrix_entries`

  Elements of the DC inductance matrix (per unit length).

- `Co=Co_matrix_entries`

  Elements of the DC capacitance matrix (per unit length).

- `Ro=Ro_matrix_entries`

  Elements of the DC resistance matrix (per unit length).

- `Go=Go_matrix_entries`

  Elements of the DC conductance matrix (per unit length).

- `Rs=Rs_matrix_entries`

  Elements of the skin-effect inductance matrix (per unit length):

  $$R = R_o + (1 + i)\sqrt{f}Rs$$

- `Gd=Gd_matrix_entries`

  Elements of the dielectric-loss conductance matrix (per unit length). The frequency dependent conductance uses the parameters *Gs* and *fp* (polarization frequency: **FP** parameter). It can be modeled in two ways according to the **MULTIDEBYE** parameter:

  - One-pole debye model (**MULTIDEBYE**=0)

$$G = G_o + \frac{G_d}{2\pi} \times \frac{i2\pi f \times 2\pi fp}{i2\pi f + 2\pi fp}$$

where *fp* is the polarization frequency (**FP** parameter).

- Multi-pole debye model (**MULTIDEBYE**=1)

  By using this option, we build a complex frequency-dependent capacitance matrix:

  $$C(\omega) = C_{inf} + f(j\omega)C_d$$

  Therefore, line conductance per unit length becomes:

  $$Y(j\omega) = G_o + (j\omega)C_{inf} + (j\omega)f(j\omega)C_d$$

  where: $C_{inf} = C - \alpha G_d$ and $C_d = \beta G_d$; *C* and *Gs* are the user-defined matrices. with:

  $$\alpha = \frac{\ln\left(\frac{\omega_2^2 + \omega_0^2}{\omega_1^2 + \omega_0^2}\right)}{4\pi\left(\operatorname{atan}\left(\frac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\frac{\omega_0}{\omega_2}\right)\right)} \text{ and } \beta = \frac{\ln(10^8)}{2\pi\left(\operatorname{atan}\left(\frac{\omega_0}{\omega_1}\right) - \operatorname{atan}\left(\frac{\omega_0}{\omega_2}\right)\right)}$$

  Finally, the function $f(j\omega) = \frac{\ln\left(\frac{\omega_2 + j\omega}{\omega_1 + j\omega}\right)}{\ln(10^8)}$ which is fitted with 15 real poles.

  Note: $\omega_0 = 2\pi f_p$, $\omega_1 = 10^4$, $\omega_2 = 10^{12}$, $\omega = 2\pi f$ and *fp* the polarization frequency (**FP** parameter).

- Compat dispersive model (**COMPAT**=1)

  $$G = G_o + G_d \times \frac{f}{\sqrt{1 + \left(\frac{f}{f_{gd}}\right)^2}}$$

  where f_{gd} is a cut-off frequency; if f_{gd} value is zero then G keeps linear dependency on the frequency. Default is zero.

Example RLGC Model

Circuit name: *RLGCmodel_example.cir*.

```
.MODEL model_rlgc W MODELTYPE=RLGC N=2
+ Lo = 0.3481e-6
+ 0.5458e-7 0.3481e-6
+ Co = 0.1593e-9
+ -0.2578e-10  0.1651e-9
```

```
+ Ro = 75
+ 0 50
+ Go = 0.2421e-3
+ -0.4860e-4  0.2070e-3
+ Rs = 0.0025
+ 0.0014
+ Gd = 1.2e-13
+ -4.1e-14  1.1e-13
```

Tabular RLGC Model Syntax

The Tabular model is an extension of the RLGC model, which allows to model transmission line arbitrary frequency-dependent behavior. There is no limitation on the number of coupled lines. Inductance and capacitance tabular matrices (C_o and L_o) have to be specified, the other tabular matrices are optional. Each tabular matrix is described in a **`.MODEL`** statement.

General Instantiation of the Model

```
.MODEL model_name sp W MODELTYPE=TABLE N=nb_line
+ LMODEL=L_freq_model CMODEL=C_freq_model
+ [RMODEL=R_freq_model] [GMODEL=G_freq_model] [FITTABLEMODEL=val]
```

Parameters

- `N=nb_line`

  Number of lines.

- `LMODEL=L_freq_model`

  Name of the model containing the sampled values of the inductance matrix.

- `CMODEL=C_freq_model`

  Name of the model containing the sampled values of the capacitance matrix.

- `RMODEL=R_freq_model`

  Name of the model containing the sampled values of the resistance matrix. Default is zero.

- `GMODEL=G_freq_model`

  Name of the model containing the sampled values of the conductance matrix. Default is zero.

- `FITTABLEMODEL=val`

  When set to 1, it will enable a causal model (admittance and propagation) to be built from non-causal tabulated data. If set to 0 (default), the tabulated data will not be modified to build the model and the built-in models are considered to be causals.

Example Tablemodel

```
.model ex1 W MODELTYPE=TABLE N=2 LMODEL=lmod1
+ CMODEL=cmod1 Rmodel=rmod1 Gmodel=gmod1
```

Sampled Matrix Model

This tabular matrix model gives a frequency-varying behavior of R, L, C and G matrices.

General Instantiation of the Model

```
.MODEL model_name sp N=nb_line
+ SPACING=spacing_type VALTYPE=value_type
+ [INFINITY=matrix_values]
+ DATA=tabular_matrix_values
```

Parameters

- `model_name`

  Name of the model.

- `N=nb_line`

  Number of lines.

- `SPACING=spacing_type`

  Data spacing format: only NONUNIFORM type is handled.

- `VALTYPE=value_type`

  Type of matrix elements: only REAL type is handled.

- `INFINITY=matrix_values`

  Data points at infinity.

- `DATA=tabuled_matrix_values`

  Specified frequency value and corresponding matrix data points. As the matrices are symmetric, only the lower-half portion is described. Syntax: `DATA=(sampled_number, f1 data1 f2 data2 ...)`.

Example Tablemodel

As the model is a “two coupled transmission line”, the dimension of the matrices is 2. Therefore, on each line of the `DATA` specification, after the sample number, the first value is the frequency, the second is the (1,1) diagonal value, the third is the (2,1) off-diagonal value, and the last is the (2,2) diagonal value.

```
.model cmod1 sp N=2 SPACING=NONUNIFORM VALTYPE=REAL
+ DATA=(1,( 6.602360e-11 -7.04724e-12 6.602360e-11))

.MODEL lmod1 sp N=2 SPACING=NONUNIFORM VALTYPE=REAL
+ INFINITY=(4.0076e-7 4.6030e-8 4.0076e-7)
+ DATA=( 20,
+ (0.000000e+00 3.934460e-07 4.6030e-08 3.933460e-07)
+ (3.746488e+06 4.151139e-07 4.6030e-08 4.151959e-07)
+ (7.726980e+06 4.084730e-07 4.6030e-08 4.085604e-07)
+ (1.196411e+07 4.054831e-07 4.6030e-08 4.055730e-07)
+ (1.648352e+07 4.037715e-07 4.6030e-08 4.037628e-07)
```

```
+ (3.204884e+07 4.008228e-07 4.6030e-08 4.008166e-07)
+ (5.911330e+07 3.988513e-07 4.6030e-08 3.988467e-07)
+ (7.650809e+07 3.981851e-07 4.6030e-08 3.981811e-07)
+ (8.650875e+07 3.978968e-07 4.6030e-08 3.978931e-07)
+ (9.756098e+07 3.976313e-07 4.6030e-08 3.976278e-07)
+ (1.098398e+08 3.973847e-07 4.6030e-08 3.973813e-07)
+ (1.235615e+08 3.971538e-07 4.6030e-08 3.971507e-07)
+ (2.962963e+08 3.958050e-07 4.6030e-08 3.958030e-07)
+ (3.428571e+08 3.956319e-07 4.6030e-08 3.956300e-07)
+ (4.010283e+08 3.954596e-07 4.6030e-08 3.954579e-07)
+ (5.753425e+08 3.951106e-07 4.6030e-08 3.951092e-07)
+ (7.145791e+08 3.949294e-07 4.6030e-08 3.949281e-07)
+ (9.230769e+08 3.947392e-07 4.6030e-08 3.947380e-07)
+ (1.269625e+09 3.945339e-07 4.6030e-08 3.945329e-07)
+ (4.000000e+09 3.940153e-07 4.6030e-08 3.940147e-07)
+ )

.MODEL rmod1 sp N=2 SPACING=NONUNIFORM VALTYPE=REAL
+ DATA=( 18,
+ (0.000000e+00 8.765530e-01 6.299210e-03 8.765530e-01)
+ (3.746488e+06 6.028640e+00 6.299210e-03 6.028640e+00)
+ (7.726980e+06 8.270684e+00 6.299210e-03 8.270684e+00)
+ (1.196411e+07 1.007694e+01 6.299210e-03 1.007694e+01)
+ (2.131439e+07 1.313620e+01 6.299210e-03 1.313620e+01)
+ (3.803487e+07 1.727973e+01 6.299210e-03 1.727973e+01)
+ (6.741573e+07 2.271433e+01 6.299210e-03 2.271433e+01)
+ (7.650809e+07 2.414031e+01 6.299210e-03 2.414031e+01)
+ (9.756098e+07 2.714662e+01 6.299210e-03 2.714662e+01)
+ (1.098398e+08 2.845071e+01 6.299210e-03 2.845071e+01)
+ (1.764706e+08 3.620734e+01 6.299210e-03 3.620734e+01)
+ (1.995249e+08 3.844427e+01 6.299210e-03 3.844427e+01)
+ (2.264151e+08 4.085570e+01 6.299210e-03 4.085570e+01)
+ (3.428571e+08 5.012232e+01 6.299210e-03 5.012232e+01)
+ (4.010283e+08 5.413628e+01 6.299210e-03 5.413628e+01)
+ (7.145791e+08 7.197077e+01 6.299210e-03 7.197077e+01)
+ (1.269625e+09 9.584070e+01 6.299210e-03 9.584070e+01)
+ (4.000000e+09 1.690795e+02 6.299210e-03 1.690795e+02)
+ )

.MODEL gmod1 sp N=2 SPACING=NONUNIFORM VALTYPE=REAL
+ DATA=( 22,
+ (0.000000e+00 5.977166e-11 0.000000e+00  5.977166e-11)
+ (3.746488e+06 1.451137e-05 -1.821096e-06 1.451043e-05)
+ (7.726980e+06 2.992905e-05 -3.755938e-06 2.992712e-05)
+ (1.196411e+07 4.634076e-05 -5.815525e-06 4.633777e-05)
+ (2.131439e+07 8.245729e-05 -1.036052e-05 8.245196e-05)
+ (3.803487e+07 1.473209e-04 -1.848803e-05 1.473114e-04)
+ (5.911330e+07 2.289642e-04 -2.873385e-05 2.289494e-04)
+ (6.741573e+07 2.611221e-04 -3.276951e-05 2.611062e-04)
+ (7.650809e+07 2.963396e-04 -3.718913e-05 2.963205e-04)
+ (9.756098e+07 3.778840e-04 -4.742254e-05 3.778596e-04)
+ (1.098398e+08 4.253437e-04 -5.339105e-05 4.254163e-04)
+ (1.389961e+08 5.383752e-04 -6.756338e-05 5.383404e-04)
+ (1.564859e+08 6.061286e-04 -7.606484e-05 6.060795e-04)
+ (1.995249e+08 7.728220e-04 -9.698528e-05 7.727721e-04)
+ (2.264151e+08 8.769759e-04 -1.100561e-04 8.769193e-04)
+ (2.962963e+08 1.146647e-03 -1.440240e-04 1.147553e-03)
+ (3.428571e+08 1.327992e-03 -1.666563e-04 1.327906e-03)
```

```
+ (4.757709e+08 1.842808e-03 -2.312632e-04 1.842689e-03)
+ (5.753425e+08 2.228580e-03 -2.796630e-04 2.228336e-03)
+ (9.230769e+08 3.575363e-03 -4.486902e-04 3.575132e-03)
+ (1.959184e+09 7.588526e-03 -9.523220e-04 7.588036e-03)
+ (4.000000e+09 1.549424e-02 -1.944324e-03 1.549234e-02)
+ )
```

Error Message Treatment

Most of the errors you can meet with this model are the same as for the LDTL model, see "Error Message Treatment" on page 172.

Also, the following error message is displayed if there is a lack of value(s) in the C matrix description:

```
ERROR IN RLGC FILE 2lin.rlc : check matrix C
```

Lossy Transmission Line: U Model

```
Uxx P1...PN PGNDin PN+1...P2N PGNDout UNAME L=length [SAVEFIT=val]
```

The U model is implemented in Eldo to simulate lossy-coupled uniform lines. This model can be used in all analysis modes (DC, AC, Transient, SST, SSTNOISE, or MODSST). Examples are provided in the directory *$MGC_AMS_HOME/examples/tlines/U_model*.

Parameters

- `xx`

  Transmission line name.

- `P1…PN`

  The N nodes at one end of the line system for a system consisting of N lines.

- `PGNDin`

  Reference node for the P1…PN nodes of the line system.

- `PN+1…P2N`

  The N nodes at the other end of the line system. The line number i in the line system connects the nodes Pi and PN+i.

- `PGNDout`

  Reference node for the PN+1…P2N nodes of the line system.

- `UNAME`

  Name of the lossy transmission line model.

- `L=length`

  Geometric length of the system (meter). Default value is 1.0. If `L`=0, Eldo uses the default value.

- `SAVEFIT=val`

  If the value is 1, this option saves the initialization of the transmission line model (in the file *circuit_name.fit*), in order to speed up the following simulations of the same netlist. Default value is 0.

Example

Circuit name: *Umodel_elev1_example.cir*.

```
U1 1 0 2 0 Umodel L=1.0e-3
```

Specifies a lossy transmission line `U1` between nodes 1 and 2, the reference plane is the ground (node 0). The length of this transmission line is 1.0e-3 and all the parameters are specified in the model called `Umodel` (**`.MODEL U`** model).

Model Syntax

```
.MODEL UNAME U LEVEL=3 ELEV=elev_val PLEV=plev_val
+ [DEV=dlev_val] [LLEV=llev_val] [Param=p_val]
```

Parameters

- `UNAME`

  Name of the model.

- `LEVEL=3`

  Selects the model of lossy transmission line.

- `ELEV=elev_val`

  Selects the specification format:

  `ELEV=1` → geometrical description.

  `ELEV=2` → precomputed model parameters (R, L, C, and G matrices).

  `ELEV=3` → measured parameters.

- `PLEV=plev_val`

  Selects the type of transmission line: planar structure (**`PLEV`**=1), coax (**`PLEV`**=2) or twinhead (**`PLEV`**=3). Only planar structure is supported.

- `DLEV=dlev_val`

  Specifies the dielectric and ground reference configuration. Two configurations are proposed: microstrip layered dielectric (**`DLEV`**=1) and stripline (**`DLEV`**=2). Default value is 1.

- `LLEV=llev_val`

  Reference plane inductance consideration (default is 0):

  `LLEV=0` → omit this inductance.

  `LLEV=1` → include this inductance (not supported).

- `Param=p_val`

  Specifies parameters of the lines (depends on the specification format).

Geometric Description: ELEV=1

Restriction

Only single line can be described.

Specific Parameters

- `DLEV`

  Type of Line;

  `DLEV=1` → Microstrip layered dielectric

`DLEV=2` → Stripline

- `NL`

  Number of line, default value is 1. (only single line can be described).

- `HT`

  Conductor height, default value is 2.0e-4m.

- `WD`

  Conductor width, default value is 3.0e-4m.

- `TH`

  Conductor thickness, default value is 1.0e-4m.

- `KD`

  Dielectric relative permittivity, default value is 10.0.

- `RHO`

  Conductor resistivity. Default value is 17e-9 Ωm (copper).

Example

Circuit name: *Umodel_elev1_example.cir*.

```
.MODEL Umodel U LEVEL=3 ELEV=1 PLEV=1 DLEV=2 NL=1
+ HT=1.0e-4 + WD=2.0e-4 TH=5.0e-5 RHO=1.785e-8
```

This model describes a lossy stripline.

Precomputed Model Parameters: ELEV=2

The precomputed parameters correspond to the R, L, C and G matrices. Since these matrices are symmetric, only the upper-triangular parts are specified.

Restriction

This description allows the specification of up to five signal conductors.

Specific Parameters

- `crj`

  Self capacitance per unit length (Fm^{-1}). Default value is $1.0e^{-9}$.

- `cij`

  Mutual capacitance per unit length (Fm^{-1}). Default value is 0.

- `ljj`

  Self inductance per unit length (Hm^{-1}). Default value is $1.0e^{-6}$.

- `lij`

  Mutual inductance per unit length (Hm^{-1}). Default value is 0.

- `rjj`

  Resistance per unit length (Ωm^{-1}). Default value is 0.

- `grj`

  Self conductance per unit length (Sm^{-1}). Default value is 0.

- `gij`

  Mutual conductance per unit length (Sm^{-1}). Default value is 0.

Example

Circuit name: *Umodel_elev2_example.cir*.

```
.MODEL Umodel U LEVEL=3 ELEV=2 PLEV=1 r11=34.48
+ r22=34.48 + r33=34.48 l11=49.76n l22=49.76n l33=49.76n
+ l12=7.65n + l23=7.65n cr1=10.82p cr2=11.24p cr3=10.82p
+ c12=-1.97p + c23=-1.97p gr1=0.15u gr2=0.15u gr3=0.15u
```

This model describes three coupled lossy transmission lines.

Measured Parameters: ELEV=3

This description corresponds to the electrical parameters.

Restriction

Only single line can be described.

Specific Parameters

- `ZK`

  Characteristic impedance (Ω).

- `VREL`

  Relative velocity.

- `DELAY`

  Delay(s) for length **`DELEN`**.

- `CAPL`

  Linear capacitance in length **`CLEN`**. Default value is 1.

- `AT1`

  Attenuation factor in length **`ATLEN`**. Default value is 1.

- `DELEN`

  Unit of length (m) for **`DELAY`**. Default value is 1.

- `CLEN`

  Unit of length (m) for **`CAPL`**. Default value is 1.

- `ATLEN`

  Unit of length for **`AT1`**. Default value is 1.

- `FR1`

  Frequency at which dispersion starts (only affects resistance). If no value is specified, the dispersion will not be taken into account.

In order to discard redundant parameter sets, the following equations are used:

Table 4-12. Lossy Transmission Line: U Model Parameter Combinations

Input Parameters	Computation
ZK, DELAY, DELEN, CAPL and CLEN	Redundant, discard CAPL and CLEN
ZK, VREL, CAPL and CLEN	Redundant, discard CAPL and CLEN
ZK, DELAY and DLEN	$VREL = \frac{DLEN}{DELAY \times CLIGHT}$
ZK and VREL	$C = \frac{1.0}{ZK \times VREL \times CLIGHT}$ $L = \frac{ZK}{VREL \times CLIGHT}$
ZK, CAPL and CLEN	$C = \frac{CAPL}{CLEN}$ $L = C \times ZK^2$
CAPL, CLEN, DELAY and DELEN	$VREL = \frac{DELEN}{DELAY \times CLIGHT}$
CAPL, CLEN and VREL	$C = \frac{CAPL}{CLEN}$ $L = \frac{1.0}{C \times VREL^2 \times CLIGHT^2}$

Example

Circuit name: *Umodel_elev3_example.cir*.

```
.MODEL Umodel U LEVEL=3 ELEV=3 PLEV=1 ZK=50 DELAY=10n AT1=1
```

This model describes a single lossy transmission line with a characteristic impedance of 50 Ω.

Error Message Treatment

Most of the errors you can meet with this model are the same as for the LDTL model, see “Error Message Treatment” on page 172.

Microstrip Models

A set of microstrip and stripline layout discontinuity structures is provided. This set is targeting RF simulations where a piece of microstrip or stripline discontinuity has to be included. They may also be used for integrated design of microstrip or stripline structures. The available set is as follows:

Microstrip Discontinuities:

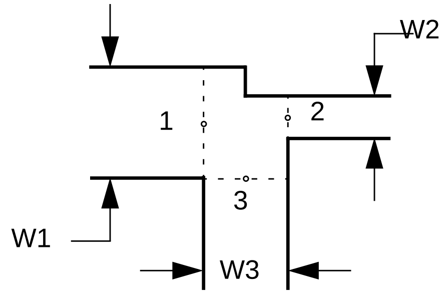
- MTEE—Microstrip T Junction
- MBEND—Microstrip Bend (Arbitrary Angle, Optimally Mitered)
- MBEND2—90-degree Microstrip Bend (Mitered)
- MBEND3—90-degree Microstrip Bend (Optimally Mitered)
- MCORN—90-degree Microstrip Bend (Unmitered)
- MSTEP—Microstrip Step in Width
- VIA2—Cylindrical Via Hole in Microstrip

Stripline Discontinuities:

- SBEND—Unmitered Stripline Bend
- STEE—Stripline T Junction
- SSTEP—Stripline Step in Width

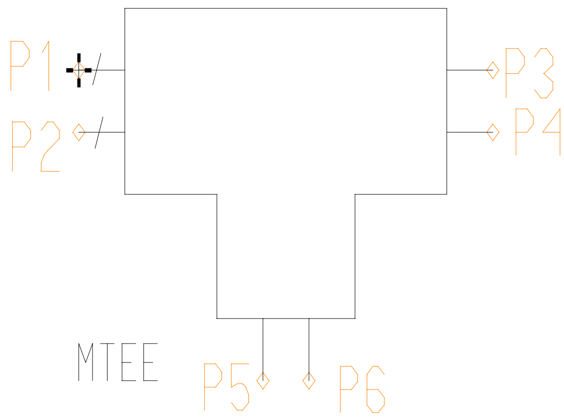
MTEE—Microstrip T Junction

Figure 4-6. Microstrip T Junction



Symbol

Figure 4-7. Microstrip T Junction Symbol



Syntax

```
Yxx MTEE P1 P2 P3 P4 P5 P6 PARAM: [W1=val] [W2=val] [W3=val]
+ [T=val] [Er=val] [H=val]
```

Parameters

Table 4-13. Microstrip T Junction Parameters

Parameter	Definition	Default	Units
W1	Conductor width of the first arm	2.0e-3	meter
W2	Conductor width of the second arm	2.0e-3	meter
W3	Conductor width of the third arm	3.0e-3	meter
T	Conductor thickness	5.0e-6	meter
ER	Dielectric relative permittivity	4	-
H	Dielectric thickness	1.6e-3	meter

Model Validity Range

$0.5 \le W1/H \le 2.0$

$0.5 \le W2/H \le 2.0$

$0.5 \le W3/H \le 2.0$

Simulation Domains

DC, AC, TRANSIENT, and SST

References

Brian C. Wadell, “Transmission Line Design Handbook”, 1991 Artech House.

Notes

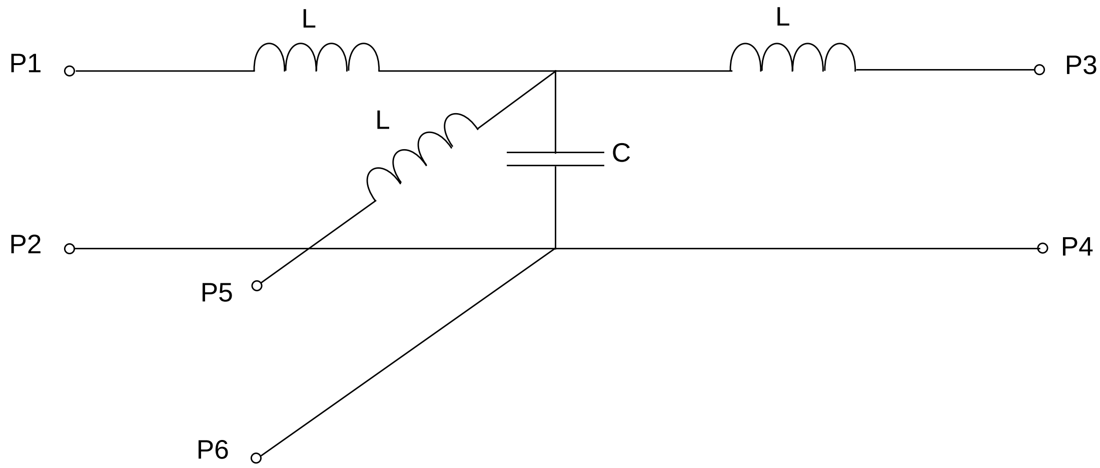
The model is based on the microstrip line symmetric T junction equations given in the mentioned reference.

The model handles symmetrical T-junction only. If the specified W1 and W2 parameters are not identical, the geometrical mean of W1 and W2 parameters is computed and used.

$$W1 = W2 = \sqrt{W1 \cdot W2}$$ for non-symmetrical T-junction

Figure 4-8 illustrates the model equivalent circuit and pins connections:

Figure 4-8. Equivalent circuit Microstrip T Junction



Example

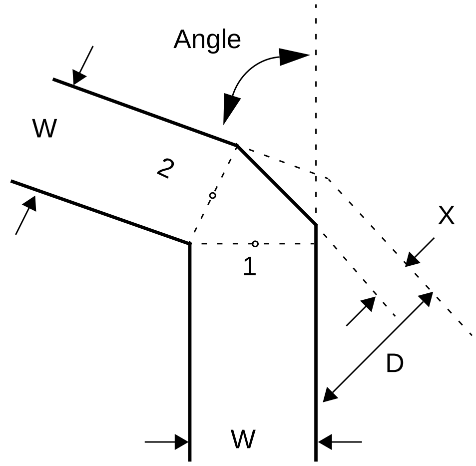
A simple MTEE s-parameter extraction example over a range of frequencies:

```
.param w1        = 2.0e-3
.param w2        = 2.0e-3
.param w3        = 3.0e-3
.param t         = 5.0e-6
.param Er        = 4
```

```
.param h         = 1.6e-3
.param frequency = 5e9

Ymtee MTEE t1a 0 t1b 0 t2 0 PARAM: W1=w1 W2=w2 W3=w3 T=t Er=Er + H=h

*** S-Parameters Extraction

V1a  t1a  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
V1b  t1b  0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90
V2   t2   0 IPORT=3 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param frequency 1e9 7e9 100e6

.sst fund1=frequency nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),frequency)
.extract fsst label=S12_Mag yval(SM(1,2),frequency)
.extract fsst label=S13_Mag yval(SM(1,3),frequency)
.extract fsst label=S21_Mag yval(SM(2,1),frequency)
.extract fsst label=S22_Mag yval(SM(2,2),frequency)
.extract fsst label=S23_Mag yval(SM(2,3),frequency)
.extract fsst label=S31_Mag yval(SM(3,1),frequency)
.extract fsst label=S32_Mag yval(SM(3,2),frequency)
.extract fsst label=S33_Mag yval(SM(3,3),frequency)

.end
```

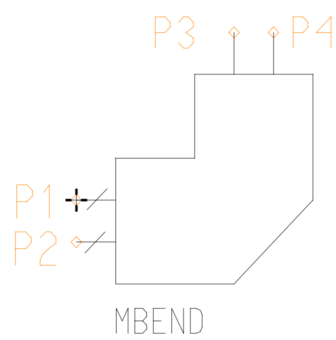
MBEND—Microstrip Bend (Arbitrary Angle, Optimally Mitered)

Figure 4-9. Microstrip Bend (Arbitrary Angle, Optimally Mitered)



Symbol

Figure 4-10. Microstrip Bend (Arbitrary Angle, Optimally Mitered) Symbol



Syntax

```
Yxx MBEND P1 P2 P3 P4 PARAM: [W=val] [H=val] [Er=val] [T=val]
+ [RHO=val] [TAND=val] [M=val] [ANGLE=val]
```

Parameters

Table 4-14. Microstrip Bend (Arbitrary Angle, Optimally Mitered) Parameters

Parameter	Definition	Default	Units
W	Conductor width	2.0e-3	meter
H	Dielectric thickness	1.6e-3	meter
ER	Dielectric relative permittivity	4	-

Table 4-14. Microstrip Bend (Arbitrary Angle, Optimally Mitered) Parameters

Parameter	Definition	Default	Units
T	Conductor thickness	5.0e-6	meter
RHO	Conductor resistivity	1.7e-8	Ohm.meter
TAND	Dielectric loss tangent	0.0	-
M	Optimal mitre percentage	60	%
ANGLE	Bend angle	60	degree

Model Validity Range

$1 \le ER \le 128$

$0 < ANGLE < 90$

$0.01 \le W/H \le 100$

Simulation Domains

DC, AC, TRANSIENT, and SST

References

Brian C. Wadell, “Transmission Line Design Handbook”, 1991 Artech House.

Equations

$$M = \frac{100\ X}{d} \qquad (\%)$$

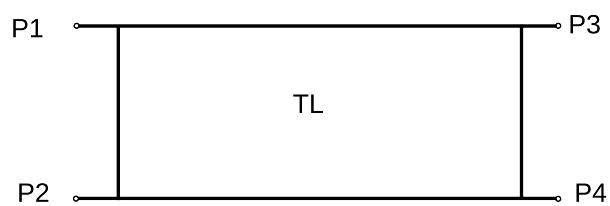
$$d - X = \sqrt{2}\ W\left(1 - \frac{M}{100}\right)$$

The model is equivalent to a transmission line of length:

$$l = \frac{2\ M}{100\ \sin(ANGLE)}$$

Figure 4-11 illustrates the model equivalent circuit and pins connections:

Figure 4-11. Equivalent circuit Microstrip Bend (Arbitrary Angle, Optimally Mitered)



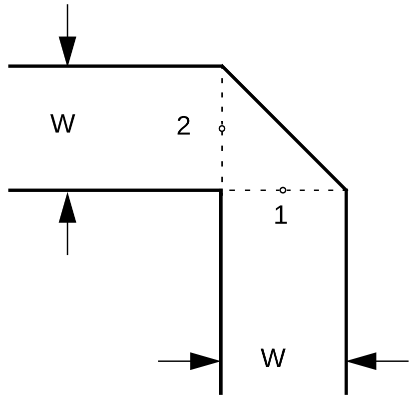
Example

A simple MBEND s-parameter extraction example over a range of frequencies:

```
.param W     = 2.0e-3
.param H     = 1.6e-3
.param Er    = 4
.param T     = 5.0e-6
.param RHO   = 1.7e-8
.param TAND  = 0
.param M     = 60
.param angle = 60
.param fx    = 5e9

Ymbend MBEND in 0 out 0 PARAM: W=W H=H Er=Er T=T RHO=RHO
+ TAND=TAND M=M ANGLE=ANGLE

*** S-Parameters Extraction

Vin  in  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
Vout out 0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param fx 1e9 7e9 100e6

.sst fund1=fx nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),fx)
.extract fsst label=S12_Mag yval(SM(1,2),fx)

.end
```

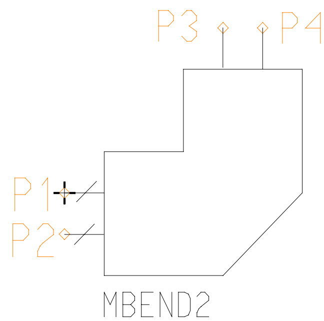
MBEND2—90-degree Microstrip Bend (Mitered)

Figure 4-12. 90-degree Microstrip Bend (Mitered)



Symbol

Figure 4-13. 90-degree Microstrip Bend (Mitered) Symbol



Syntax

```
Yxx MBEND2 P1 P2 P3 P4 PARAM: [H=val] [W=val] [Er=val]
```

Parameters

Table 4-15. 90-degree Microstrip Bend (Mitered) Parameters

Parameter	Definition	Default	Units
H	Substrate thickness	1.6e-3	meter
W	Conductor width	2.0e-3	meter
ER	Dielectric constant	4	-

Model Validity Range

$0.2 < W/H < 6$

$2.36 < ER < 10.4$

Simulation frequency < 12/H (Frequency in GHz, H in mm)

Simulation Domains

DC, AC, TRANSIENT, and SST

References

M. Kirschning, R. H. Jansen, and N. H. L. Koster. "Measurement and Computer-Aided Modeling of Microstrip Discontinuities by an Improved Resonator Method," 1983 IEEE MTT-S International Microwave Symposium Digest, May 1983, pp.495-497.

Equations

The equivalent circuit of the MBEND2 consists of two inductors and a capacitor, shown in Figure 4-14.

Equations used to calculate the equivalent circuit component values:

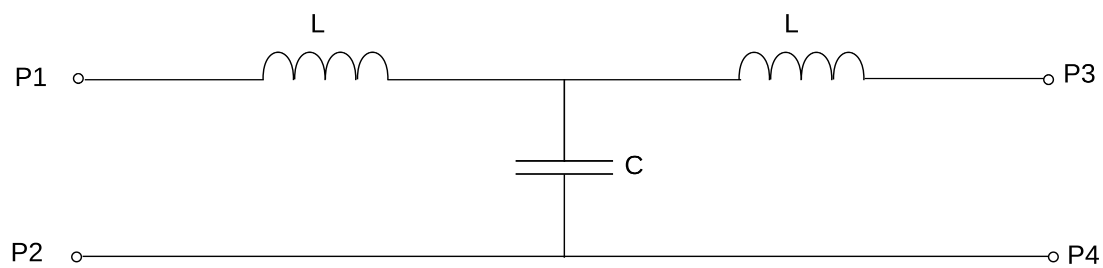
$$\frac{C}{H} = \frac{W}{H}\left[7.6\ \mathrm{Er}\ + 3.8 + \frac{W}{H}(3.93\ \mathrm{Er}\ + 0.62)\right] \qquad \frac{pF}{m}$$

$$\frac{L}{H} = 441.2712\left\{1 - 1.062\exp\left[-0.177\left(\frac{W}{H}\right)^{0.947}\right]\right\} \qquad \frac{pF}{m}$$

Notes

The model parameters validity ranges were tested at the corners and some typical design values.

Figure 4-14. Equivalent circuit MBEND2



Example

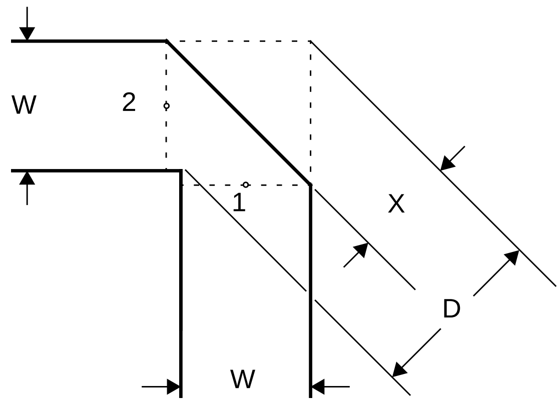
A simple MBEND2 s-parameter extraction example over a range of frequencies:

```
.param H  = 1.6e-3
.param W  = 2.0e-3
.param Er = 4
.param fx = 1e9

Ymbend2 MBEND2 in 0 out 0 PARAM: H=H W=W Er=Er

*** S-Parameters Extraction
```

```
Vin  in  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
Vout out 0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param fx 1e9 7e9 100e6

.sst fund1=fx nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),fx)
.extract fsst label=S12_Mag yval(SM(1,2),fx)

.end
```

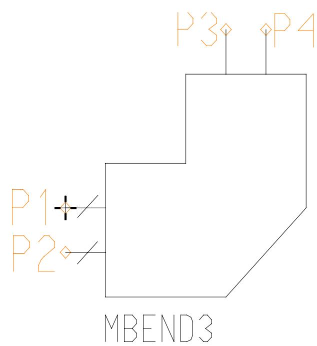
MBEND3—90-degree Microstrip Bend (Optimally Mitered)

Figure 4-15. 90-degree Microstrip Bend (Optimally Mitered)



Symbol

Figure 4-16. 90-degree Microstrip Bend (Optimally Mitered) Symbol



Syntax

```
Yxx MBEND3 P1 P2 P3 P4 PARAM: [W=val] [H=val] [Er=val] [T=val]
+ [RHO=val] [TAND=val]
```

Parameters

Table 4-16. 90-degree Microstrip Bend (Optimally Mitered) Parameters

Parameter	Definition	Default	Units
W	Conductor width	2.0e-3	meter
H	Dielectric thickness	1.6e-3	meter
ER	Dielectric relative permittivity	4	-
T	Conductor thickness	5.0e-6	meter
RHO	Conductor resistivity	1.7e-8	Ohm.meter

Table 4-16. 90-degree Microstrip Bend (Optimally Mitered) Parameters

Parameter	Definition	Default	Units
TAND	Dielectric loss tangent	0.0	-

Model Validity Range

$0.25 \le W/H \le 2.75$

$2.5 \le ER \le 25$

Simulation frequency < 15/h (Frequency in GHz, H in mm)

Simulation Domains

DC, AC, TRANSIENT, and SST

References

Brian C. Wadell, “Transmission Line Design Handbook”, 1991 Artech House.

Equations

The optimal miter is given by:

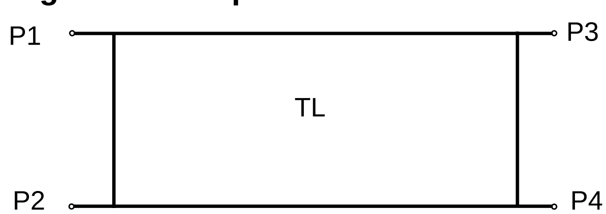
$$M = 52 + 65e^{-1.35\left(\frac{W}{H}\right)} = \frac{100\ X}{d} \qquad (\%)$$

and is modeled as a transmission line of length:

$$L = W\left[1.04 + 1.3e^{-1.35\left(\frac{W}{H}\right)}\right] \qquad m$$

The following figure illustrates the model equivalent circuit and pins connections:

Figure 4-17. Equivalent Circuit MBEND3



Example

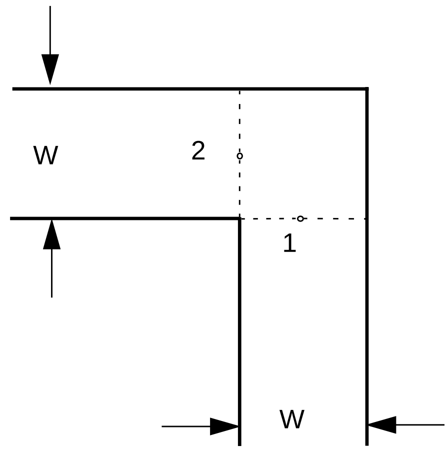
A simple MBEND3 s-parameter extraction example over a range of frequencies:

```
.param W    = 2.0e-3
.param H    = 1.6e-3
.param Er   = 4
```

```
.param T    = 5.0e-6
.param RHO  = 1.7e-8
.param TAND = 0
.param fx   = 5e9

Ymbend3 MBEND3 in 0 out 0 PARAM: W=W H=H Er=Er T=T RHO=RHO
+ TAND=TAND

*** S-Parameters Extraction

Vin  in  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
Vout out 0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param fx 1e9 7e9 100e6

.sst fund1=fx nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),fx)
.extract fsst label=S12_Mag yval(SM(1,2),fx)

.end
```

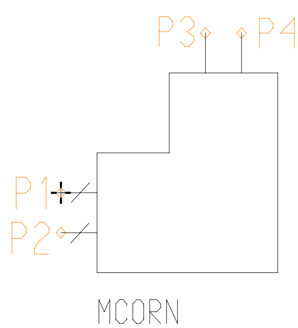
MCORN—90-degree Microstrip Bend (Unmitered)

Figure 4-18. 90-degree Microstrip Bend (Unmitered)



Symbol

Figure 4-19. 90-degree Microstrip Bend (Unmitered) Symbol



Syntax

```
Yxx MCORN P1 P2 P3 P4 PARAM: [W=val] [H=val] [Er=val]
```

Parameters

Table 4-17. 90-degree Microstrip Bend (Unmitered) Parameters

Parameter	Definition	Default	Units
H	Substrate thickness	1.6e-3	meter
W	Conductor width	2.0e-3	meter
ER	Dielectric constant	4	-

Model Validity Range

$0.1 \leq W/H \leq 6$

$2 \leq ER \leq 15$

Simulation Domains

DC, AC, TRANSIENT, and SST

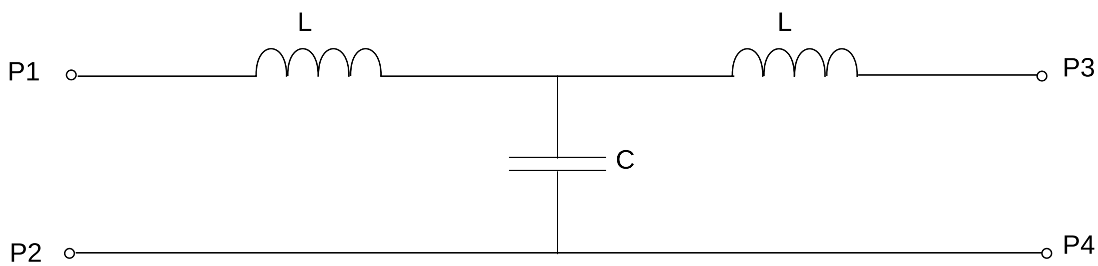
References

M. Kirschning, R. H. Jansen, and N. H. L. Koster. "Measurement and Computer-Aided Modeling of Microstrip Discontinuities by an Improved Resonator Method," 1983 IEEE MTT-S International Microwave Symposium Digest, May 1983, pp. 495-497.

Equations

The equivalent circuit of a microstrip corner is a lumped network of two inductors and a capacitor, as shown in Figure 4-20.

Figure 4-20. Equivalent circuit Microstrip corner



The following equations are used to calculate the values of the model lumped components:

$$L = \left(1 - 1.35 \times e^{\left\{-0.18\left(\frac{W}{H}\right)^{1.39}\right\}}\right) \times 0.2 \quad nH$$

$$C = \left\{(10.35\ \mathrm{Er} + 0.25)\left(\frac{W}{H}\right)^2 + (2.6\ \mathrm{Er} + 5.44)\left(\frac{W}{H}\right)\right\} 0.001 \times H \qquad pF$$

where H is in mm.

Example

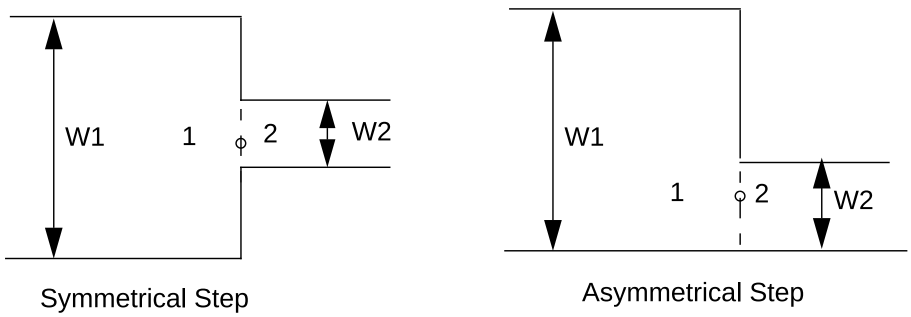
A simple MCORN s-parameter extraction example over a range of frequencies:

```
.param H  = 1.6e-3
.param W  = 2.0e-3
```

```
.param Er = 4
.param fx = 5e9

Ymcorn MCORN in 0 out 0 PARAM: H=H W=W Er=Er

*** S-Parameters Extraction

Vin  in  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
Vout out 0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param fx 1e9 7e9 100e6

.sst fund1=fx nharm1=1

.extract fsst label=S11_Mag_Eldo yval(SM(1,1),fx)
.extract fsst label=S12_Mag_Eldo yval(SM(1,2),fx)

.end
```

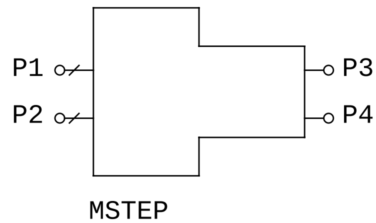
MSTEP—Microstrip Step in Width

Figure 4-21. Microstrip Step in Width



Symbol

Figure 4-22. Microstrip Step in Width Symbol



Syntax

```
Yxx MSTEP P1 P2 P3 P4 PARAM: [W1=val] [W2=val] [ER=val]
+ [H=val] [F=val] [ASYMMETRICAL=val] [T=val]
```

Parameters

Table 4-18. Microstrip Step in Width Parameters

Parameter	Definition	Default	Units
W1	Conductor width at port 1	2.0e-3	meter
W2	Conductor width at port 2	0.5e-3	meter
H	Substrate thickness	1.6e-3	meter
ER	Relative Dielectric constant	4	-
T	Conductor thickness	5.0e-6	meter
ASYMMETRICAL[1]	Selects between symmetrical and asymmetrical step structures	0	-
F	Operating frequency	1e9	Hz

1. ASYMMETRICAL only takes two values, 1 for asymmetrical, and 0 for symmetrical step.

Simulation Domains

DC, AC, TRANSIENT, and SST

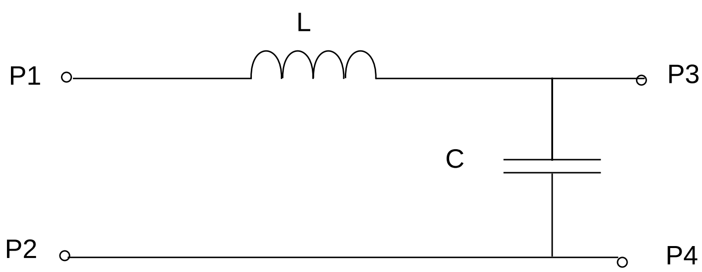
References

Brian C. Wadell, “Transmission Line Design Handbook”, 1991 Artech House.

Equations

The equivalent circuit of a microstrip step is a lumped network of an inductor and a capacitor, as shown in Figure 4-23.

Figure 4-23. Equivalent circuit of a microstrip step in width



The model equations to calculate the lumped component values are given in the mentioned reference.

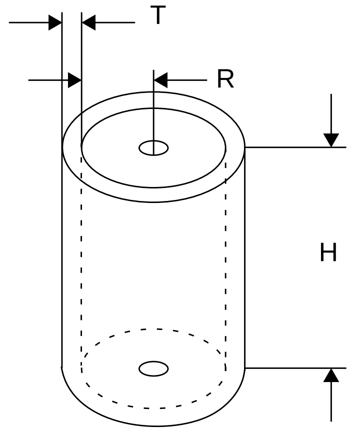
Example

A simple MSTEP s-parameter extraction example over a range of frequencies:

```
.param W1        = 2.0e-3
.param W2        = 0.5e-3
.param H         = 1.6e-3
.param ER        = 4
.param Frequency = 1e9
.param T         = 5.0e-6

Ymstep MSTEP t1a 0 t1b 0 PARAM: W1=W1 W2=W2 ER=ER H=H F=Frequency
ASYMMETRICAL=0 T=T

*** S-Parameters Extraction

V1a  t1a  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
V1b  t1b  0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param Frequency 1e9 5e9 100e6

.sst fund1=Frequency nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),Frequency)
.extract fsst label=S12_Mag yval(SM(1,2),Frequency)

.end
```

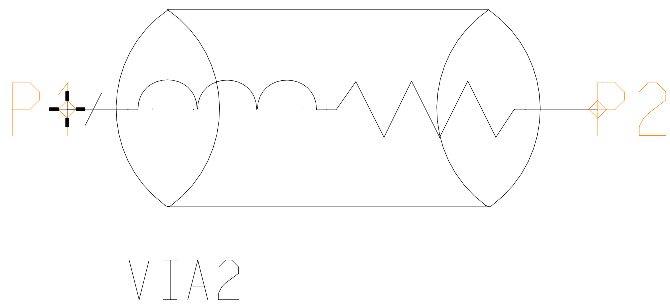
VIA2—Cylindrical Via Hole in Microstrip

Figure 4-24. Cylindrical Via Hole in Microstrip



Symbol

Figure 4-25. Cylindrical Via Hole in Microstrip Symbol



Syntax

```
Yxx VIA2 P1 P2 PARAM: [H=val] [R=val] [COND=val] [T=val] [F=val]
```

Parameters

Table 4-19. Cylindrical Via Hole in Microstrip Parameters

Parameter	Definition	Default	Units
H	Substrate thickness	200.0e-6	meter
R	Via radius	100.0e-6	meter
COND	Conductor conductivity	58.842e6	1/(Ohm.meter)
T	Conductor thickness	5.0e-6	meter
F	Operating center frequency	1.0e9	Hz

Model Validity Range

100um < H < 635um

0.1 < R/H < 1.5

0 < T < R

Simulation Domains

DC, AC, TRANSIENT, and SST

References

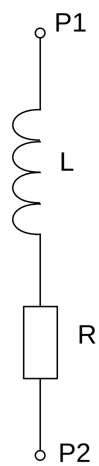
M. Goldfarb and R. Pucel. "Modeling Via Hole Grounds in Microstrip," IEEE Microwave and Guided Wave Letters, Vol. 1, No. 6, June, pp.135-137.

Notes

The VIA2 is modeled as a series resistor and inductor network. The resistor and inductor values are based on equations given in the mentioned reference.

Figure 4-26 illustrates the model equivalent circuit and pins connections:

Figure 4-26. Equivalent circuit Cylindrical Via Hole in Microstrip



Example

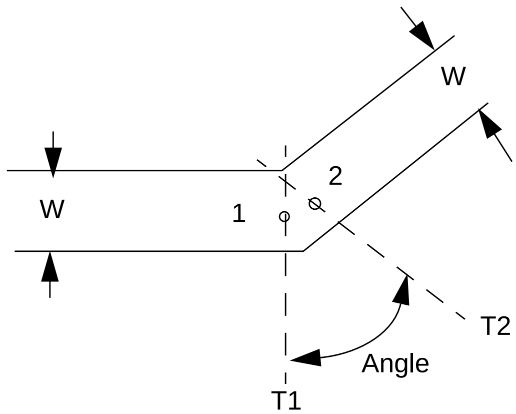
A simple VIA2 s-parameter extraction example over a range of frequencies:

```
.param H    = 200e-6
.param R    = 100e-6
.param COND = 58.824e6
.param T    = 5.0e-6
.param fx   = 5e9

Yvia2 VIA2 in out PARAM: H=H R=R COND=COND T=T F=fx

*** S-Parameters Extraction
```

```
Vin  in  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
Vout out 0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param fx 1e9 7e9 100e6

.sst fund1=fx nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),fx)
.extract fsst label=S12_Mag yval(SM(1,2),fx)

.end
```

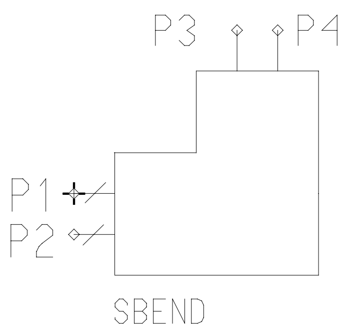
SBEND—Unmitered Stripline Bend

Figure 4-27. Unmitered Stripline Bend



Symbol

Figure 4-28. Unmitered Stripline Bend Symbol



Syntax

```
Yxx SBEND P1 P2 P3 P4 PARAM: [W=val] [B=val] [ER=val] [T=val]
+ [ANGLE=val] [F=val]
```

Parameters

Table 4-20. Unmitered Stripline Bend Parameters

Parameter	Definition	Default	Units
W	Conductor width	0.1e-3	meter
B	Ground plane spacing	280e-6	meter
T	Conductor thickness	17e-6	meter
ER	Relative dielectric constant	4.2	-
ANGLE	Bend angle	60	degree

Table 4-20. Unmitered Stripline Bend Parameters

Parameter	Definition	Default	Units
F	Simulation frequency	2e9	Hz

Simulation Domains

DC, AC, TRANSIENT, and SST

References

Altschuler, H.M., and A.A. Oliner, “Discontinuities in the Center Conductor of Symmetric Strip Transmission Line,” IRE Transactions on Microwave Theory and Techniques, Vol. MTT-8, May 1960, pp. 328-339 and “Addendum to ‘Discontinuities in the Center Conductor of Symmetric Strip Transmission Line’,” Vol. MTT-10, No. 2, March 1962, p. 143.

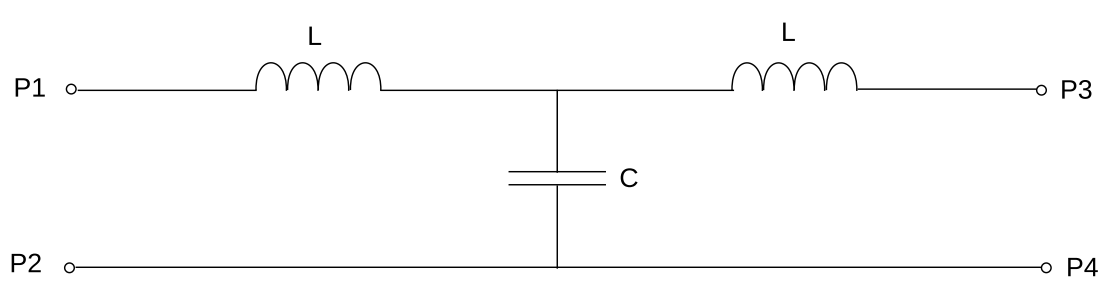
K.C. Gupta, “Computer-Aided Design of Microwave Circuits”, 1981, ARTECH HOUSE, INC.

Arthur A. Oliner, “Equivalent Circuits for Discontinuities in balanced Strip Transmission Line”, Microwave Theory and Techniques, IEEE Transactions on, Volume: 3 Issue: 2, Mar 1955.

Notes

The equivalent circuit of an unmitered stripline bend is a lumped network of two inductors and a capacitor whose values are based on equations given in the mentioned references. The equivalent circuit is shown in Figure 4-29.

Figure 4-29. Equivalent circuit of an unmitered stripline bend



Example

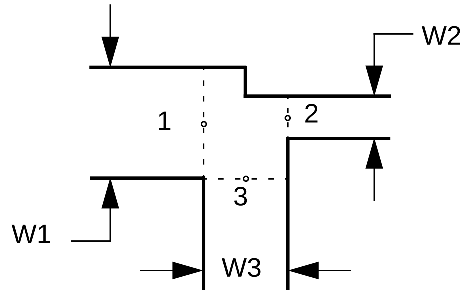
A simple SBEND s-parameter extraction example over a range of frequencies:

```
.param W     = 0.1e-3
.param B     = 280e-6
.param ER    = 4.2
.param T     = 17e-6
.param ANGLE = 60
.param F     = 2e9
```

```
Ysbend SBEND in 0 out 0 PARAM: W=W B=B ER=ER T=T ANGLE=ANGLE F=F

*** S-Parameters Extraction

Vin  in  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
Vout out 0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param f 1e9 5e9 100e6

.sst fund1=f nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),f)
.extract fsst label=S12_Mag yval(SM(1,2),f)

.end
```

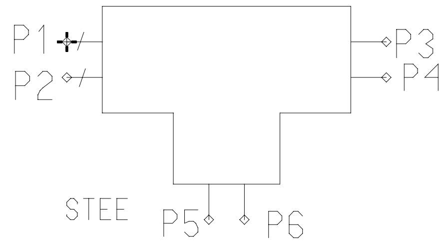
STEE—Stripline T Junction

Figure 4-30. Stripline T Junction



Symbol

Figure 4-31. Stripline T Junction Symbol



Syntax

```
Yxx STEE P1 P2 P3 P4 P4 P5 PARAM: [W1=val] [W2=val] [W3=val]
+ [B=val] [ER=val] [T=val] [F=val]
```

Parameters

Table 4-21. Stripline T Junction Parameters

Parameter	Definition	Default	Units
W1	First arm conductor width	0.1e-3	meter
W2	Second arm conductor width	0.1e-3	meter
W3	Third arm conductor width	0.2e-3	meter
B	Ground plane spacing	280e-6	meter
T	Conductor thickness	17e-6	meter
ER	Relative dielectric constant	4.2	-
F	Simulation frequency	2e9	Hz

Simulation Domains

DC, AC, TRANSIENT, and SST

References

Altschuler, H.M., and A.A. Oliner, “Discontinuities in the Center Conductor of Symmetric Strip Transmission Line,” IRE Transactions on Microwave Theory and Techniques, Vol. MTT-8, May 1960, pp. 328-339 and “Addendum to ‘Discontinuities in the Center Conductor of Symmetric Strip Transmission Line’,” Vol. MTT-10, No. 2, March 1962, p. 143.

K.C. Gupta, “Computer-Aided Design of Microwave Circuits”, 1981, ARTECH HOUSE, INC.

Arthur A. Oliner, “Equivalent Circuits for Discontinuities in balanced Strip Transmission Line”, Microwave Theory and Techniques, IEEE Transactions on, Volume: 3 Issue: 2, Mar 1955.

Notes

The model is based on the microstrip line symmetric T junction equations given in the mentioned references.

The model handles symmetrical T-junction only. If the specified W1 and W2 parameters are not identical, the geometrical mean of W1 and W2 parameters is computed and used.

$$W1 = W2 = \sqrt{W1 \cdot W2} \text{ for non-symmetrical T-junction}$$

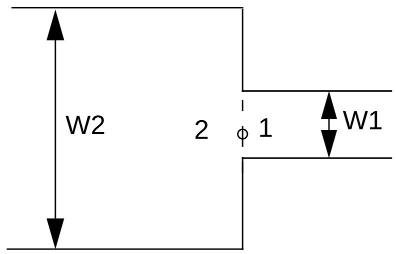
Example

A simple STEE s-parameter extraction example over a range of frequencies:

```
.param W1        = 0.1e-3
.param W2        = 0.1e-3
.param W3        = 0.2e-3
.param B         = 280e-6
.param ER        = 4.2
.param T         = 17e-6
.param frequency = 2e9

Ystee STEE t1a 0 t1b 0 t2 0 PARAM: W1=W1 W2=W2 W3=W3 B=B ER=ER T=T
F=frequency

*** S-Parameters Extraction

V1a  t1a  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
V1b  t1b  0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90
V2   t2   0 IPORT=3 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param frequency 1e9 5e9 100e6

.sst fund1=frequency nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),frequency)
.extract fsst label=S12_Mag yval(SM(1,2),frequency)
.extract fsst label=S13_Mag yval(SM(1,3),frequency)
.extract fsst label=S33_Mag yval(SM(3,3),frequency)

.end
```

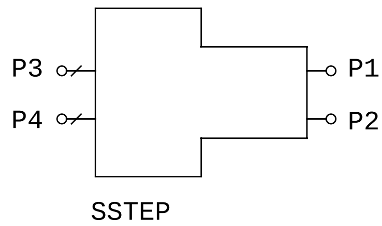
SSTEP—Stripline Step in Width

Figure 4-32. Stripline Step in Width



Symbol

Figure 4-33. Stripline Step in Width Symbol



Syntax

```
Yxx SSTEP P1 P2 P3 P4 PARAM: [W1=val] [W2=val] [B=val] [T=val]
+ [ER=val] [F=val]
```

Parameters

Table 4-22. Stripline Step in Width Parameters

Parameter	Definition	Default	Units
W1	Conductor width at port 1	0.10e-3	meter
W2	Conductor width at port 2	0.15e-3	meter
B	Ground plane spacing	280e-6	meter
T	Conductor thickness	17e-6	meter
ER	Relative dielectric constant	4.2	-
F	Simulation frequency	1e9	Hz

Simulation Domains

DC, AC, TRANSIENT, and SST

References

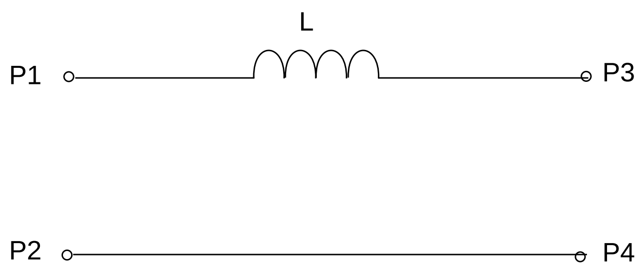
Brian C. Wadell, “Transmission Line Design Handbook”, 1991 Artech House.

Notes

The SSTEP is modeled as a series inductor. The inductor value is based on equations given in the mentioned reference.

Figure 4-34 illustrates the model equivalent circuit and pins connections:

Figure 4-34. Equivalent circuit for a Stripline Step in Width



Example

A simple SSTEP s-parameter extraction example over a range of frequencies:

```
.param W1         = 0.10e-3
.param W2         = 0.15e-3
.param B          = 280e-6
.param T          = 17e-6
.param ER         = 4.2
.param frequency  = 1e9

Ysstep SSTEP t1a 0 t1b 0 PARAM: W1=W1 W2=W2 B=B T=T ER=ER F=frequency

*** S-Parameters Extraction

V1a  t1a  0 IPORT=1 RPORT=50 FOUR fund1 PdBm (1) -100 -90
V1b  t1b  0 IPORT=2 RPORT=50 FOUR fund1 PdBm (1) -100 -90

.step param frequency 1e9 5e9 100e6

.sst fund1=frequency nharm1=1

.extract fsst label=S11_Mag yval(SM(1,1),frequency)
.extract fsst label=S12_Mag yval(SM(1,2),frequency)

.end
```

Junction Diode

```
Dxx NP NN [NM] MNAME [[AREA=]AREA_VAL] [PERI|PJ|PD=PERIVAL]
+ [PGATE=PGATE_VAL] [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [OFF=0|1]
+ [STATISTICAL=0|1] [NOISE=0|1] [NONOISE]
Dxx NP NN [NM] MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

Parameters

- `xx`

  Junction diode name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `NM`

  Name of the `M` node. Any number of pins can be specified. However, only some proprietary models will make use of additional pins, if any.

- `MNAME`

  Name of the model used, as described in a **`.MODEL`** command.

- **`AREA`**`=AREA_VAL`

  Model area factor. Default value is 1.

- **`PERI|PJ|PD`**`=PERIVAL`

  Perimeter of the diode. Default value is 0. **`PERI`**, **`PJ`**, and **`PD`** keywords are all synonymous.

- **`PGATE`**`=PGATE_VAL`

  Length of the diode gate-edge. Default value is 0 (used for JUNCAP model only).

- **`T`**`=TVAL`

  Sets temperature for the individual diode. Default nominal temperature is 27 °C.

- **`M`**`=VAL`

  Device multiplier, simulating the effect of multiple devices in parallel. All currents, capacitances and resistances are affected by **`M`**. Default value is 1.

  The device is first evaluated without the **`M`** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **`M`**. Input values **`W`** and **`L`** are not affected. Models are chosen depending on input **`W`** and **`L`**, if required. Options **`MINL`**, **`MAXL`**, **`MINW`**, **`MAXW`**, and so on, do not apply either, since they check the input values of **`W`** and **`L`**.

Note

Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **T**[**EMP**]=VAL

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **DTEMP**=VAL

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **OFF**=0|1

  When set to `1`, causes no initial operating point to be calculated for the device during DC analysis, i.e. the device is "off". When set to `0`, the option is ignored.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1. Cannot be specified after **NONOISE** keyword parameter.

- **NONOISE**

  Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the device presents no noise contribution to the noise analysis.

- **NOISE**=0|1

  When set to `0`, is equivalent to specifying the **NONOISE** flag. When set to `1`, specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **NONOISE** specification on a **.MODEL** card.

- **FMIN**=VAL

  Lower limit of the noise frequency band.

- **FMAX**=VAL

  Upper limit of the noise frequency band.

- **NBF**=VAL

  Specifies the number of sinusoidal sources with appropriated amplitude and frequency and with randomly distributed phase from which the noise source is composed. Default value is 50. This parameter has no effect when **FMIN** is set to 0.

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion. **FMIN** is also used to specify the algorithm used to generate the noise source generated by the diode. When **FMIN**>0 the diode noise source is generated with sinusoids; when **FMIN**=0 it is generated with a continuous spectrum between **FMIN** and **FMAX**.

Handling Floating Gates

When **.OPTION FLOATGATECHECK** is set, will enable Eldo to issue a warning when a floating gate is detected and resume the simulation. Eldo will not consider reverse-biased diodes as active elements when checking for floating gates. **.OPTION FLOATGATERR** can be set to enable Eldo to issue an error when a floating gate is detected and stop the simulation. In addition Eldo can force detected floating gates to 0 using **.OPTION FLOATGATE0**, which can be useful to change the topology of a circuit to achieve better convergence.

For more information please refer to “FLOATGATECHECK” on page 977.

Combining Identical Diodes

Multiple identical diodes connected in parallel are reduced into a single instance using the **M** parameter, for example:

```
d1 1 2 diode1
d2 1 2 diode1
d3 1 3 diode1
```

Here, diode instances `d1` and `d2` will be replaced by:

```
d1 1 2 diode1 M = 2
```

but `d3` will remain as it is because it is connected to different nodes.

Note

It will only work when no parameters are given on the instance.

For more information see “Merging Devices in Parallel” on page 120.

Noise in Diodes

Noise models are available for diodes, see Noise Equations for All Levels of the *Eldo Device Equations Manual*.

Example

```
d1 1 2 diode1 T=50
```

Specifies a diode d1 placed between nodes 1 and 2 of model name diode1, and at temperature of 50°C.

Diode Model Syntax

```
.MODEL MNAME D [PAR=VAL]
```

Eldo provides several predefined diode models. The **LEVEL** parameter is placed first in the parameter list and specifies the model to be used. The options are listed in the table below. Parameters specific to the selected model are then assigned values following the **LEVEL** parameter. Default diode level is 1.

Table 4-23. Diode Models

LEVEL Value	Model Name
1	Berkeley Level 1 (Eldo Level 1)
2	Modified Berkeley Level 1 (Eldo Level 2)
3	Fowler-Nordheim Model (Eldo Level 3)
4	STMicroelectronics LEVEL 1
5	STMicroelectronics LEVEL 2
6	STMicroelectronics LEVEL 3
8[1]	JUNCAP (Eldo Level 8) (DIOLEV≠9)
8[2]	JUNCAP2 (Eldo Level 8, DIOLEV=11)
9	Philips Diode Level 500 (Eldo Level 9)
21	Diode Level 21

1. Diode LEVEL 8 has a selector (**DIOLEV**) which when set to **DIOLEV**=9, the JUNCAP1 model is used.
2. Diode LEVEL 8 has a selector (**DIOLEV**) which when set to **DIOLEV**=11, the JUNCAP2 model is used.

Please refer to the Diode Model Equations of the *Eldo Device Equations Manual*.

Using -compat with Diodes

Using the **`-compat`** runtime flag or selecting **`.option compat`** has the following effect on Diode Level values.

Table 4-24. Diode Models with -compat

LEVEL	Model Name
2	Fowler-Nordheim (Eldo LEVEL 3)
3	Berkeley Level 1 (Eldo LEVEL 1), by default SCALEV is set to 3
4	JUNCAP Diode Model (Eldo LEVEL 8), by default DIOLEV is set to 9

Output Quantities

For a listing of generic output quantities for diode models see "Diode Plotting and Printing" on page 809. This lists the syntax used to plot or print the values of both extrinsic and intrinsic pins. The information is divided into subsections of output type and analysis type.

Berkeley Level 1 (Eldo Level 1)

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 non-linear 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`**, and the saturation current temperature exponent. Reverse breakdown is modeled by an exponential increase in the reverse diode current and is determined by the parameters **`BV`** and **`IBV`** (both of which are positive numbers).

For model parameters, please refer to the Berkeley Level 1 Model (Eldo Level 1) Parameters of the *Eldo Device Equations Manual*.

Modified Berkeley Level 1 (Eldo Level 2)

The Eldo Level 2 model is based on the Berkeley diode model. It includes modified calculations for the reverse breakdown effects, the recombination effect and for the temperature behavior.

For model parameters, please refer to the Modified Berkeley Level 1 Model (Eldo Level 2) Parameters of the *Eldo Device Equations Manual*.

Fowler-Nordheim Model (Eldo Level 3)

Fowler-Nordheim diode models are formed as a metal-insulator-semiconductor device or as a semiconductor-insulator-semiconductor layer device. The insulator is sufficiently thin to permit the tunneling of carriers. This element models electrically-alterable memory cells, air-gap switches, and other insulation breakdown devices.

For model parameters, please refer to the Fowler-Nordheim Model (Eldo Level 3) Parameters of the *Eldo Device Equations Manual*.

Example

```
*DIODE model definition
.model dio d level=3
...
*main circuit
d1 2 10 dio
```

Specifies the diode d1 placed between the nodes 2 and 10 of model name dio. Default parameter values are used for this Fowler-Nordheim model.

```
*DIODE model definition
.model diode1 d rs=4.68 bv=6.10 cjo=346p
+ tt=50n m=0.33 vj=0.75 is=1e-11 n=1.27
+ ibv=20ma
...
*main circuit
dbridge 2 10 diode1
```

Specifies the diode dbridge placed between nodes 2 and 10. The electrical parameters are defined in the diode1 model.

Note

LEVEL is not defined and so the Berkeley Level 1 diode model is used (**LEVEL**=1).

JUNCAP (Eldo Level 8)

The JUNCAP model is the replica of the bulk-source/drain parasitic diode model included in the common MOS structure. This model may be used to define the bulk diode as an external component or to define new elements such as the substrate diode with similar behaviors.

LEVEL 8 has a selector (**DIOLEV**) which when set to **DIOLEV**=9 means the JUNCAP1 model is used.

For model parameters, please refer to the JUNCAP Model (Eldo Level 8) Parameters of the *Eldo Device Equations Manual*.

JUNCAP2 (Eldo Level 8, DIOLEV=11)

The JUNCAP2 model (LEVEL=8, DIOLEV=11) is the evolution of the standard JUNCAP1 junction diode model (LEVEL=8, DIOLEV=9).

For model parameters, please refer to the JUNCAP2 Model (Eldo Level 8, DIOLEV=11) Parameters of the *Eldo Device Equations Manual*.

Philips Diode Level 500 (Eldo Level 9)

The Philips Diode Level 500 model provides a detailed description of the diode currents in forward and reverse biased Si-diodes. It is meant to be used for DC, transient and AC analysis.

For model parameters, please refer to the Philips Diode Level 500 Model (Eldo Level 9) Parameters of the *Eldo Device Equations Manual*.

Diode Level 21

Level 21 is a combination of the Level 9 current equations and Level 1 capacitance model with a mix in the temperature dependencies of both models.

For model parameters, please refer to the Diode Model Level 21 Parameters of the *Eldo Device Equations Manual*.

BJT—Bipolar Junction Transistor

```
Qxx NC NB NE [NS] [TH] MNAME [[AREA=]AREA_VAL] [AREAB=AREA_VAL]
+ [AREAC=AREA_VAL] [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [OFF=0|1]
+ [STATISTICAL=0|1] [NOISE=0|1] [NONOISE]
Qxx NC NB NE [NS] MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

Parameters

- `xx`

  Name of the bipolar junction transistor.

- `NC`

  Name of the collector node.

- `NB`

  Name of the base node.

- `NE`

  Name of the emitter node.

- `MNAME`

  Name of the model used, as described in a **.MODEL** command.

- `NS`

  Substrate node. Default value is ground. This node can be specified in the **.MODEL** card as **BULK**=<node_name>.

- `TH`

  Thermal node, available only for the HICUM model (level 9). Voltage in this node represents the temperature increase of the device due to self-heating.

- **AREA**=AREA_VAL

  Relative device area. Default value is 1.

- **AREAB**=AREA_VAL

  Base relative device area. Default is **AREA**.

- **AREAC**=AREA_VAL

  Collector relative device area. Default is **AREA**.

- **T**=VAL

  Sets temperature for the individual BJT. Default value is the current simulation temperature.

- **T**[**EMP**]=VAL

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **DTEMP**=VAL

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **M**=VAL

  Device multiplier, simulating the effect of multiple devices in parallel. All currents, capacitances and resistances are affected by **M**. Default value is 1.

  The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**, **MAXL**, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note

Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **OFF**=0|1

  When set to 1, causes no initial operating point to be calculated for the device during DC analysis, i.e. the device is "off". When set to 0, the option is ignored.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- **NONOISE**

  Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the device presents no noise contribution to the noise analysis.

- **NOISE**=0|1

  When set to 0, is equivalent to specifying the **NONOISE** flag. When set to 1, specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **NONOISE** specification on a **.MODEL** card.

- **FMIN**=VAL

  Lower limit of the noise frequency band.

- **FMAX**=VAL

  Upper limit of the noise frequency band.

- **NBF**=VAL

  Specifies the number of sinusoidal sources with appropriated amplitude and frequency and with randomly distributed phase from which the noise source is composed. Default value is 50. This parameter has no effect when **FMIN** is set to 0.

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion. **FMIN** is also used to specify the algorithm used to generate the noise source generated by the BJT. When **FMIN**>0 the BJT noise source is generated with sinusoids; when **FMIN**=0 it is generated with a continuous spectrum between **FMIN** and **FMAX**.

Example

```
q1 1 2 3 bipol1 t=50
```

Specifies a bipolar junction transistor q1 connected to nodes 1, 2 and 3, of model type bipol1, at a temperature of 50°C.

Combining Identical BJTs

Multiple identical BJTs connected in parallel are reduced into a single instance using the **M** parameter, for example:

```
q1 1 2 3 bipol1
q2 1 2 3 bipol1
q3 1 2 4 bipol1
```

Here, `q3` will remain as it is because the nodes are different, but BJT instances `q1` and `q2` will be replaced by:

```
q1 1 2 3 bipol1 M=2
```

Note

This will also work even if the **AREA** parameter is specified. If two BJTs are merged and in addition have the same connectivity, the models will also have the same **AREA**.

For more information see “Merging Devices in Parallel” on page 120.

BJT Model Syntax

```
.MODEL MNAME NPN SUBS=VAL [PAR=VAL]
.MODEL MNAME PNP SUBS=VAL [PAR=VAL]
.MODEL MNAME LPNP SUBS=VAL [PAR=VAL]
```

Both NPN and PNP model types are to be used for vertical structures. This means that most of the current flows in a vertical direction. With regard to a silicon integrated transistor, this flow can also be parallel to the surface of the silicon. This type of BJT is also called a lateral transistor. It is possible in Eldo for a **PNP** BJT model to be declared as lateral by using the **LPNP** keyword in the **.MODEL** command. The current flow can also be affected by the **SUBS** parameter as follows:

If **SUBS**=-1, BJT is lateral.
If **SUBS**=1, BJT is vertical.

The **LEVEL** parameter is used to determine which model is to be used, as shown by the table below:

Table 4-25. BJT Models

LEVEL	**Model Name**
1	Modified Gummel-Poon Model (Eldo Level 1) (Berkeley Standard Model)
2	STMicroelectronics LEVEL 1
3	BNR-HICUM Model
4	Philips Mextram 503.2 Model (Eldo Level 4)
5	Improved Berkeley Model (Eldo Level 5)
7	ROCK-HICUM Model
8	VBIC v1.2 Model (Eldo Level 8)
	(**VERSION**=1.15) VBIC v1.1.5 Model (Eldo Level 8)
9	HICUM Model (Eldo Level 9)
22	Philips Mextram 504 Model (Eldo Level 22)
23	Philips Modella Model (Eldo Level 23)
24	HICUM Level0 Model (Eldo Level 24)

Using -compat with BJT Models

Using the `-compat` runtime flag or selecting **`.OPTION COMPAT`** has the following effect on BJT Level values.

Table 4-26. BJT Models with -compat

LEVEL	Model Name
2	Improved Berkeley Model (Eldo LEVEL 5)
2	STMicroelectronics LEVEL 1 (Eldo LEVEL 2)
4	VBIC v1.2 (Eldo LEVEL 8)
	(**VERSION**=1.15) VBIC v1.1.5 (Eldo LEVEL 8)
6	Philips Mextram 503.2 Model (Eldo LEVEL 4)
8	HICUM Model (Eldo LEVEL 9)

If `-compat` is set, NPN are vertical, PNP are lateral.

Output Quantities

For a listing of generic output quantities for BJT models see "BJT Plotting and Printing" on page 806. This lists the syntax used to plot or print the values of both extrinsic and intrinsic pins. The information is divided into subsections of output type and analysis type.

Noise in BJTs

Noise models are available for BJTs Level 1 and Level 5 only, see the Noise sections of the *Eldo Device Equations Manual* in BJT Level 1 and BJT Level 5 respectively. Other BJT models have their own separate noise models.

Automatic Selection of BJT Model Via AREA Specifications

BJT model versions can be selected via **`.MODEL`** command **`AREAMIN`** and **`AREAMAX`** parameters. Whenever Eldo finds a BJT device for which the model name has no **`.MODEL`** command, it searches through all defined models for a model of the same root name and whose **`AREAMIN`**/**`AREAMAX`** range matches the specified device size.

The BJT model is selected if the instance parameter **`AREA`** is consistent with **`AREAMIN`**/**`AREAMAX`** of the **`.MODEL`** command.

The separator in the **`.MODEL`** command should be a "." or a "_" character. If these characters exist multiple times in the model name, the last one specified is used as the separator. Examples for `<root>.<extension>`:

```
Q1 vplus in out 0 QND_model area=10
.model QND_model_2 NPN AREAMIN=3 AREAMAX=20
```

Here, <root> = QND_model, separator = _, <extension> = 2.

```
Q1 vplus in out 0 QND.model area=10
.model QND.model_3 NPN AREAMIN=3 AREAMAX=20
```

Here, <root> = QND.model, separator = _, <extension> = 3.

Example

```
Q1 C B S QND AREA = 10
.MODEL QND.1 NPN AREAMIN=0 AREAMAX=5
.MODEL QND.2 NPN AREAMIN=0 AREAMAX=20
```

In this example, the model selected for `Q1` will be `QND.2`.

Modified Gummel-Poon Model (Eldo Level 1)

The bipolar junction transistor model in Eldo 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 less complex Ebers-Moll model when using default values. Parameter names used in the modified Gummel-Poon model have been chosen to be more easily understood by the program user, and to better reflect both physical and circuit design thinking.

Please refer to the BJT Level 1 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Modified Gummel-Poon Model (Eldo Level 1) Parameters of the *Eldo Device Equations Manual*.

Philips Mextram 503.2 Model (Eldo Level 4)

This is the implementation of the Philips Mextram Bipolar Model 503.2 in Eldo. The basis of this work is the unclassified report 006/94 published by Philips Nat.lab., June 1995. In this model, the current through the epilayer is an explicit and continuous function of the internal and external base-collector junction voltages. The model covers all possible modes of operation such as ohmic current flow, saturated current flow and base push out both in the forward and reverse mode of operation.

Please refer to the Mextram Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Philips Mextram 503.2 Model (Eldo Level 4) Parameters of the *Eldo Device Equations Manual*.

Improved Berkeley Model (Eldo Level 5)

The improved Berkeley model is based on the Level 1 model (Gummel Poon). It includes several additional effects such as:

- Variable RC resistance due to velocity saturation.
- Quasi-saturation model based on a publication from G.M. Kull et al.
- Additional temperature effects.
- Variable exponent for high current roll off.

Please refer to the BJT Level 5 Equations of the *Eldo Device Equations Manual*. For model parameters, please refer to the Improved Berkeley Model (Eldo Level 5) Parameters of the *Eldo Device Equations Manual*.

Example

```
*BJT model definition
.model qmod npn bf=160 rb=100 cjs=2p
+ tf=0.3n tr=6n cje=3p cjc=2p vaf=100
...
*main circuit
q23 10 24 13 qmod
```

Specifies the bipolar transistor q23, with the collector connected to node 10, base to node 24 and emitter to node 13. Electrical parameters are specified in the model qmod.

VBIC v1.2 Model (Eldo Level 8)

The VBIC model is a Bipolar Junction Transistor (BJT) model. VBIC stands for **V**ertical **B**ipolar **I**ntercompany **M**odel. The VBIC model was developed as an industry-standard, public domain replacement for the SPICE Gummel-Poon (SGP) model. VBIC is designed to be as similar as possible to the SGP model, yet overcomes its major deficiencies. VBIC improvements on SGP:

- 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

- Electro-thermal modeling.

Eldo recognizes two versions of the VBIC model. To select each model the **VERSION** parameter must be used as shown below:

Table 4-27. VBIC Version Selection

Parameter Value	VBIC Version
VERSION=1.2	VBIC v1.2 (default)
VERSION=1.15	VBIC v1.1.5

The different model parameters are described in Parameter List for v1.2 and Parameter List for v1.1.5.

Please refer to the VBIC Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the VBIC v1.2 Model (Eldo Level 8) Parameters of the *Eldo Device Equations Manual*.

VBIC v1.1.5 Model (Eldo Level 8)

Note
v1.1.5 of the VBIC model was formerly Level 21 in Eldo.

Eldo recognizes two versions of the VBIC model. To select each model the **VERSION** parameter must be used as shown above. For information on Version 1.2 see "VBIC v1.2 Model (Eldo Level 8)" on page 238

Please refer to the VBIC Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the VBIC v1.1.5 Model (Eldo Level 8) Parameters of the *Eldo Device Equations Manual*.

HICUM Model (Eldo Level 9)

HICUM is a semi-physical compact bipolar transistor model. Semi-physical means that for arbitrary transistor configurations, defined by emitter size as well as number and location of base, emitter and collector fingers (or contacts, respectively), a complete set of model parameters can be calculated from a single set of technology-specific electrical and technological data. For this, the value of each element in the equivalent circuit is related to a function describing the dependence on so-called specific electrical data (such as sheet resistances and capacitances per unit area or length), technological data (such as width and doping of the collector region underneath the emitter), physical data (like mobilities), transistor dimensions (such as design rules), operating point and temperature. The availability of such a semi-physical compact model is an important precondition for circuit optimization with respect to, for example, maximum speed and low power consumption, as well as for including process variations in the design.

The name HICUM was derived from *HI*gh-*CU*rrent *M*odel, indicating that HICUM was initially developed with special emphasis on modeling the operating region at high current densities, which is very important for certain high-speed applications. Later, formulas for the calculation of the base resistance were developed, which include three-dimensional effects occurring in short transistors with an emitter length approaching the emitter width.

When the model parameter VERSION is set to a value of 2.2, 2.21, 2.22 or 2.23, some new equations are used. When set to 2.1, the old equations are used.

Table 4-28. HICUM version selection

Parameter Value	HICUM Version
VERSION=2.23	HICUM 2.23 (Default)
VERSION=2.22	HICUM 2.22
VERSION=2.21	HICUM 2.21
VERSION=2.2	HICUM 2.2 (equivalent to HICUM 2.20)
VERSION=2.1	HICUM 2.1

For HICUM v2.1, please refer to the HICUM v2.1 Equations of the *Eldo Device Equations Manual*. For HICUM v2.1 model parameters, please refer to the HICUM v2.1 Model (Eldo Level 9) Parameters of the *Eldo Device Equations Manual*.

For HICUM v2.2, please refer to the HICUM v2.2 Equations of the *Eldo Device Equations Manual*. For HICUM v2.2 model parameters, please refer to the HICUM v2.2 Model (Eldo Level 9) Parameters of the *Eldo Device Equations Manual*.

Philips Mextram 504 Model (Eldo Level 22)

This is the implementation of the Philips Mextram Bipolar Model 504 in Eldo. The basis of this work is the unclassified report NL-UR 2000/811 published by Philips Nat.Lab. The Mextram 504 model is implemented in Eldo as LEVEL=22.

Mextram 504 has two main improvements with respect to Mextram 503. The description of the currents and changes is much smoother as a function of bias, and the parameter extraction has been improved.

Four versions, Mextram 504, Mextram 504.1, Mextram 504.6.1 and Mextram 504.7, are accessible through the model parameter VERSION. By default, Mextram 504.7 (VERSION=504.7) is selected.

Please refer to the Mextram 504 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Philips Mextram 504 Model (Eldo Level 22) Parameters of the *Eldo Device Equations Manual*.

Printing/Plotting States

If a state is to be monitored by the user, the user has to type in the netlist for a given transistor `Q1` to monitor, for example `In`:

`.PLOT DC S(Q1->In)` for DC or
`.PLOT AC S(Q1->In)` for AC or
`.PLOT TRAN S(Q1->In)` for TRAN

Philips Modella Model (Eldo Level 23)

The Modella (*Model lat*eral) model (Eldo Level = 23) developed by Philips provides an accurate model dedicated to lateral PNP devices. This new model is based on a totally new approach, accounting for the complex bi-dimensional structure of lateral transistors. The Modella model allows the simulation of lateral devices using real physically based parameters, instead of using less accurate empirically-modified vertical models, such as Gummel-Poon.

In the design of bipolar analog integrated circuits, greater flexibility is often achieved when both NPN and PNP transistors are incorporated in the circuit design. Many present day bipolar production processes use the conventional lateral PNP as the standard PNP transistor structure. For accurate modeling of such a lateral PNP transistor it is important to take the complex two-dimensional nature of the transistor into account. The physics based Modella model (*Model la*teral) does exactly this. Using a modeling approach whereby the main currents and charges are independently related to bias-dependent minority carrier concentrations. Current crowding effects, high injection effects, and a bias dependent output impedance are all taken into account.

Please refer to the Modella Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Philips Modella Model (Eldo Level 23) Parameters of the *Eldo Device Equations Manual*.

Example of a Modella Model Card

```
.MODEL modella_model LPNP LEVEL=23
+ IS= 1.8E-16 BF= 131.0 IBF= 2.6E-14 VLF= 0.54 IK= 1.1E-04
+ XIFV=0.43 EAFL= 20.5 EAFV= 75.0 BR= 25.0 IBR= 1.2E-13
+ VLR= 0.48 XIRV= 0.43 EARL= 13.1 EARV= 104.0 XES= 2.7E-3
```

```
+ XHES= 0.70 XCS= 3.0 XHCS= 1.0 ISS= 4.0E-13 RCEX= 5.0
+ RCIN= 47.0 RBCC= 10.0 RBCV= 10.0 RBEC= 10.0 RBEV= 50.0
+ REEX= 27.0 REIN= 66.0 RSB= 1.E15 TLAT= 2.4E-9 TFVR= 3.0E-8
+ TFN= 2.0E-10 CJE= 6.1E-14 VDE= 0.52 PE= 0.3 TRVR= 1.0E-9
+ TRN= 3.0E-9 CJC= 3.9E-13 VDC= 0.57 PC= 0.36 CJS= 1.3E-12
+ VDS= 0.52 PS= 0.35 TREF= 25.00 DTA= 0.0 VGEB = 1.206
+ VGCB= 1.206 VGSB= 1.206 VGB= 1.206 VGE= 1.206 VGJE= 1.123
+ AE= 4.48 SPB= 2.853 SNB= 2.6 SNBN= 0.3 SPE= 0.73 SPC= 0.73
+ SX= 1.0 KF= 1.0 AF= 1.0 EXPHI= 1
```

DC Operating Point Output

The DC operating point output facility gives information on the state of a device at its operating point. The following table shows the DC operating points that are printed in the *.chi* file in an OP and AC analysis.

Please refer to the DC Operating Point Output of the *Eldo Device Equations Manual*.

To print or plot a state for `Q1`, a BJT transistor, the following syntax is required:

- For DC

```
.PLOT DC S(Q1->GBE)
```

- For AC

```
.PLOT AC S(Q1->GBE)
```

- For Transient

```
.PLOT TRAN S(Q1->GBE)
```

HICUM Level0 Model (Eldo Level 24)

The HICUM (HIgh CUrrent bipolar compact transistor Model) Level0 model for bipolar transistors is a simplified version of HICUM v2.x with less computational effort while keeping more accurate simulation results than with the well-known Spice-Gummel-Poon-Model. The latter is caused by the more accurate transit time and transfer current formulation. However HICUM v2.x is recommended for more accurate results in physical based device modeling of bipolar transistors.

The HICUM/Level0 model is implemented in Eldo as LEVEL=24. Four versions of the HICUM/Level0 model are available in Eldo: versions v1.0, v1.11, v1.12 and v1.2. The different versions are accessible through the Eldo **VERSION** specification:

Table 4-29. HICUM/Level0 version selection

Parameter Value	HICUM/Level0 Version
VERSION=1.2	HICUM/Level0 1.2 (Default)

Table 4-29. HICUM/Level0 version selection

Parameter Value	HICUM/Level0 Version
VERSION=1.12	HICUM/Level0 1.12
VERSION=1.11	HICUM/Level0 1.11
VERSION=1.0	HICUM/Level0 1.0

Please refer to the HICUM Level0 Equations of the *Eldo Device Equations Manual*. For model parameters, please refer to the HICUM Level0 Model (Eldo Level 24) Parameters of the *Eldo Device Equations Manual*.

JFET—Junction Field Effect Transistor

```
Jxx ND NG NS MNAME [[AREA=]AREA_VAL] [L=VAL] [W=VAL]
+ [T[EMP]=VAL] [DTEMP=VAL] [M=VAL] [OFF=0|1] [STATISTICAL=0|1]
+ [NONOISE] [NOISE=1]
Jxx ND NG NS MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

Parameters

- `xx`

  JFET transistor name.

- `ND`

  Name of the drain node.

- `NG`

  Name of the gate node.

- `NS`

  Name of the source node.

- `MNAME`

  Name of the model used, as described in a **`.MODEL`** command.

Optional Parameters

- **`AREA`**

  Relative device area (optional). Default value is 1.

- **`L`**`=VAL`

  Channel length in meters.

- **`W`**`=VAL`

  Channel width in meters.

- **`T`**`[`**`EMP`**`]=VAL`

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **`DTEMP`**`=VAL`

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

`TEMP` and **`DTEMP`** are mutually exclusive. If both are specified, the last one is used.

- **M**=VAL

  Device multiplier, simulating the effect of multiple devices in parallel. All currents, capacitances and resistances are affected by **M**. Default value is 1.

  The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**, **MAXL**, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note

Using an **M** factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **OFF**=0|1

  When set to 1, causes no initial operating point to be calculated for the device during DC analysis, i.e. the device is “off”. When set to 0, the option is ignored.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- **NONOISE**

  Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the device presents no noise contribution to the noise analysis.

- **NOISE**=1

  Specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **NONOISE** specification on a **.MODEL** card.

- **FMIN**=VAL

  Lower limit of the noise frequency band.

- **FMAX**=VAL

  Upper limit of the noise frequency band.

- **NBF**=VAL

  Specifies the number of sinusoidal sources with appropriated amplitude and frequency and with randomly distributed phase from which the noise source is composed. Default value is 50. This parameter has no effect when **FMIN** is set to 0.

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion.

Note

FMIN is also used to specify the algorithm used to generate the noise source generated by the JFET. When **FMIN**>0 the JFET noise source is generated with sinusoids; when **FMIN**=0 it is generated with a continuous spectrum between **FMIN** and **FMAX**.

Noise in JFETs

Noise models are available for JFETs, see the Noise section in the JFET and MESFET Equations chapter of the *Eldo Device Equations Manual*.

Example

```
j1 1 2 3 fet2 t=50
```

Specifies a junction field effect transistor j1 placed between nodes 1, 2 and 3 of model type fet2, and at a temperature of 50°C.

JFET Model Syntax

```
.MODEL MNAME NJF [PAR=VAL]
.MODEL MNAME PJF [PAR=VAL]
```

The JFET model is derived from the FET model of Schichman & Hodges. The DC characteristics are defined by the **VTO** and **BETA** parameters, 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. The ohmic resistances, **RD** and **RS** are included. Charge Storage is modeled by non-linear depletion layer capacitances for both gate junctions. These capacitances vary with the –1/2 power of junction voltage and are defined by **CGS**, **CGD** and **PB**. The following predefined JFET device models can be selected using the **LEVEL** parameter:

Table 4-30. JFET Models

LEVEL	Model Name
1	Schichman & Hodges Model
2	Enhanced Schichman & Hodges Model
3	Extended Schichman & Hodges Model

Please refer to the JFET and MESFET Equations of the *Eldo Device Equations Manual*. For model parameters, please refer to the JFET Model (Eldo Level 1, 2 and 3) Parameters of the *Eldo Device Equations Manual*.

Example

```
*JFET model definition
.model je20 njf vto=-3.2 beta=0.98m
+ lambda=2.5m cgs=5p cgd=1.3p is=7p
...
*main circuit
j1 3 2 0 je20
```

Specifies the transistor j1 with drain connected to node 3, gate to node 2 and source to ground. The electrical parameters are specified in the model je20.

Output Quantities

For a listing of generic output quantities for JFET models see "JFET Plotting and Printing" on page 808. This lists the syntax used to plot or print the values of both extrinsic and intrinsic pins. The information is divided into subsections of output type and analysis type.

MESFET—Metal Semiconductor Field Effect Transistor

```
Jxx ND NG NS MNAME [AREA] [L=VAL] [W=VAL] [T[EMP]=VAL] [DTEMP=VAL]
+ [M=VAL] [OFF=0|1] [STATISTICAL=0|1] [NONOISE] [NOISE=1]
Jxx ND NG NS MNAME [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

See "JFET—Junction Field Effect Transistor" on page 244 for more details and the **.MODEL** syntax.

The following predefined MESFET device models can be selected using the **LEVEL** parameter:

Table 4-31. MESFET Models

LEVEL	Model Name
6	Curtice Model
7	Schichman & Hodges Model
8	Statz Model
9	TriQuint Model

Please refer to the JFET and MESFET Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the MESFET Model (Eldo Level 6, 7 and 8) Parameters of the *Eldo Device Equations Manual*.
For model parameters, please refer to the MESFET Model (Eldo Level 9, Update 1, 2 and 3) Parameters of the *Eldo Device Equations Manual*.

MOSFET

```
Mxx ND NG NS [NB] [{NN}] [MOD[EL]=]MNAME [[L=]VAL] [[W=]VAL]
+ [AD=VAL] [AS=VAL] [PD=VAL] [PS=VAL] [GEO=VAL] [NRD=VAL] [NRS=VAL]
+ [M=VAL] [RDC=VAL] [RSC=VAL] [T[EMP]=VAL] [DTEMP=VAL] [STATISTICAL=0|1]
+ [NONOISE] [NOISE=1]
Mxx ND NG NS [NB] [{NN}] [MOD[EL]=]MNAME [W=VAL] [L=VAL]
+ [FMIN=VAL] [FMAX=VAL] [NBF=VAL] [STATISTICAL=0|1]
```

Parameters

- `xx`

  MOS transistor name.

- `ND`

  Name of the drain node.

- `NG`

  Name of the gate node.

- `NS`

  Name of the source node.

- `NB`

  Name of the bulk node. If not specified, the user can specify the parameter **BULK**=node_name with the **.MODEL** command, Eldo will connect the bulk node to node_name. By default node_name is 0.

- `NN`

  Name of the N node. Any number of pins can be specified. However, only the BSIMSOI3 model and some other proprietary models will make use of additional pins, if any.

- [**MOD**[**EL**]=]MNAME

  Name of the model used, as described in a **.MODEL** command. Using the optional **MOD**[**EL**] keyword, allows the user to avoid any instantiation errors that may occur due to the MOS instantiation allowing the specification of more than four pins.

  Eldo will assume that the model name is the string that precedes the first string that is followed by an "=" sign. If there is no such string, the model name is assumed to be the 5th string, for example. In the example below, E is assumed to be the model name.

  ```
  M1 A B C D E F ...
  ```

  In the example below, F is assumed to be the model name.

  ```
  M1 A B C D E F <PNAME>=<value>
  ```

  In the example below, G is the model name because the **MOD** keyword is present.

  ```
  M1 A B C D E MOD=G ...
  ```

- **L**=VAL

  Channel length in meters. Default value is 100 m.

- **W**=VAL

  Channel width in meters. Default value is 100 m.

- **AD**=VAL

  Drain area in m^2.

- **AS**=VAL

  Source area in m^2.

- **PD**=VAL

  Drain perimeter in meters.

- **PS**=VAL

  Source perimeter in meters.

- **GEO**=VAL

  Switch which determines the layout of the device:

  GEO=0 (default value) indicates the drain and source of the device are not shared by the other devices

  GEO=1 indicates the drain is shared with another device

  GEO=2 indicates the source is shared with another device

  GEO=3 indicates the source and drain are shared with another device

- **NRD**=VAL

  Equivalent number of squares of the drain diffusion. **NRD** is multiplied with sheet resistance RSH to obtain parasitic series drain resistance of each transistor. RSH is specified in the **.MODEL** command.

- **NRS**=VAL

  Equivalent number of squares of the source diffusion. **NRS** is multiplied with sheet resistance RSH to obtain the parasitic series source resistance of each transistor.

- **M**=VAL

  This is the device “multiplier,” simulating the effect of multiple devices in parallel. Effective width, overlap, junction capacitances, and junction currents are multiplied by **M**. Parasitic resistance values are divided by **M**. Default value is 1.

  The device is first evaluated without the **M** factor, and at the very end of the device computation, all scaling quantities are multiplied / divided by **M**. Input values **W** and **L** are not affected. Models are chosen depending on input **W** and **L**, if required. Options **MINL**, **MAXL**, **MINW**, **MAXW**, and so on, do not apply either, since they check the input values of **W** and **L**.

Note
Using an M factor value less than 1 could lead to simulating devices that cannot be physically realized.

- **RDC**=VAL

  Additional drain resistance due to contact resistance.

- **RSC**=VAL

  Additional source resistance due to contact resistance.

- **T**[**EMP**]=VAL

  Sets temperature for the individual device, in degrees Celsius. Default nominal temperature=27°C.

- **DTEMP**=VAL

  Temperature difference between the device and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note
TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the device. 0 means the device will keep its nominal values. 1 means the device has statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- **NONOISE**

  Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the device presents no noise contribution to the noise analysis.

- **NOISE**=1

  Specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **NONOISE** specification on a **.MODEL** card.

- **FMIN**=VAL

  Lower limit of the noise frequency band.

- **FMAX**=VAL

  Upper limit of the noise frequency band.

- **NBF**=VAL

  Specifies the number of sinusoidal sources with appropriated amplitude and frequency and with randomly distributed phase from which the noise source is composed. Default value is 50. This parameter has no effect when **FMIN** is set to 0.

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion.

Note

FMIN is also used to specify the algorithm used to generate the noise source generated by the MOSFET. When **FMIN**>0 the MOSFET noise source is generated with sinusoids; when **FMIN**=0 it is generated with a continuous spectrum between **FMIN** and **FMAX**.

Note

Default value for all optional parameters is zero unless otherwise stated. If **L** or **W** are not specified, via the device instantiation statement, then they will take the values of the **DEFL** and **DEFW** parameters in the **.OPTION** command, see "DEFL=VAL" on page 984. The MOS geometry parameters can be assigned directly or via ".PARAM" on page 778.

Combining Identical MOSFETs

Multiple identical MOS instances connected in parallel are reduced into a single instance using the **M** parameter, for example:

```
M1 1 2 3 4 MOS1 w=10u l=3u
M2 1 2 3 4 MOS1 w=10u l=3u
M3 1 2 3 4 MOS2 w=10u l=3u
```

Here, M3 will remain because it has a different model name, but MOS instances M1 and M2 will be replaced by:

```
M1 1 2 3 4 MOS1 w=10u l=3u M=2
```

For more information see "Merging Devices in Parallel" on page 120.

MOSFET Models

```
.MODEL MNAME NMOS [PAR=VAL]
.MODEL MNAME PMOS [PAR=VAL]
```

- PAR=VAL

  Names and values of the model parameters.

The following predefined MOSFET device models can be selected using the **LEVEL** parameter. The names in parentheses are aliases.

Table 4-32. MOSFET Models

LEVEL	Model Name	Type	
		Capacitive	Charge
1 (BERK1)	Berkeley LEVEL 1 Berkeley SPICE Models	Yes	
2 (BERK2)	Berkeley LEVEL 2 Berkeley SPICE Models	**XQC**≥0.5	**XQC**<0.5
3 (BERK3)	Berkeley LEVEL 3 Berkeley SPICE Models	**XQC**≥0.5	**XQC**<0.5
4 (MERCK4)	Merckel Charge Control MERCKEL MOSFET Models		Yes
5 (MERCK1)	Merckel Simplified MERCKEL MOSFET Models	Yes	
6 (MERCK2)	Merckel Submicron MERCKEL MOSFET Models	Yes	
7 (MERCK3)	Merckel 3 (MHS) MERCKEL MOSFET Models	Yes	
8 (BSIM1)	Berkeley SPICE BSIM1 Model (Eldo Level 8)		Yes
9	reserved		
10	reserved		
11 (BSIM2)	Berkeley SPICE BSIM2 Model (Eldo Level 11)		Yes
12 (ELDO2)	Modified Berkeley Level 2 (Eldo Level 12)	**XQC**≥0.5	**XQC**<0.5
13 (ELDO3)	Modified Berkeley Level 3 (Eldo Level 13)	**XQC**≥0.5	**XQC**<0.5
14	reserved		
16 (ELDO6)	Modified Lattin-Jenkins-Grove Model (Eldo Level 16)	**XQC**≥0.5	**XQC**<0.5
17 (ELDO7)	Enhanced Berkeley Level 2 Model (Eldo Level 17)	**XQC**≥0.5	**XQC**<0.5
18 (STMOS1)	STMicroelectronics LEVEL 1		Yes
19 (STMOS3)	STMicroelectronics LEVEL 3		Yes
20 to 40	User Defined Models		
41 (STPROM1)	STMicroelectronics PROM LEVEL1		Yes
42 (STPROM3)	STMicroelectronics PROM LEVEL 3		Yes
43 (STMOS4)	STMicroelectronics MOS LEVEL 4		Yes
44 (EKV)	EKV MOS Model (Eldo Level 44 or EKV)	**XQC**≥0.5	**XQC**<0.5

Table 4-32. MOSFET Models

LEVEL	Model Name	Type	
		Capacitive	Charge
47 (BSIM3)	Berkeley SPICE BSIM3v2 Model (Eldo Level 47)		Yes
53 (BSIM3V3)	Berkeley SPICE BSIM3v3 Model (Eldo Level 53) BSIM3v3.0, BSIM3v3.1 & BSIM3v3.2		Yes
54 (SSIM)	Motorola SSIM Model (Eldo Level 54 or SSIM)		Yes
55 (BSIMSOI3)	Berkeley SPICE BSIMSOI3 v1.3 Model (Eldo Level 55)		Yes
56 (BSIMSOI3)	Berkeley SPICE BSIMSOI3 v2.x and v3.x Model (Eldo Level 56)		Yes
59 (MOSP9)	Philips MOS 9 Model (Eldo Level 59 or MOSP9)		Yes
60 (BSIM4)	Berkeley SPICE BSIM4 Model (Eldo Level 60) BSIM4.0, BSIM4.1, BSIM4.2, BSIM4.3 & BSIM4.4		Yes
61 (EKV3)	EKV3 MOS Model (Eldo Level 61 or EKV3)		Yes
62 (TFT)	TFT Polysilicon Model (Eldo Level 62) v1.0 & v2.0		Yes
63 (MOSP11)	Philips MOS Model 11 Level 1101 (Eldo Level 63)		Yes
64 (TFT)	TFT Amorphous-Si Model (Eldo Level 64)		Yes
65 (MOSP11)	Philips MOS Model 11 Level 1100 (Eldo Level 65)		Yes
66 (HISIM)	HiSIM Model (Eldo Level 66)		Yes
67 (SP)	SP Model (Eldo Level 67)		Yes
69 (M1102)	Philips MOS Model 11 Level 1102 (Eldo Level 69)		Yes
70 (PSP)	Philips PSP Model (Eldo Level 70)		Yes
71 (MM20)	Philips MOS Model 20 Level 2002 (Eldo Level 71)		Yes
72 (BSIMSOI4)	Berkeley SPICE BSIMSOI4.0 Model (Eldo Level 72)		Yes
73 (HISIMLDO)	HiSIM-LDMOS Model (Eldo Level 73)		Yes
74	MOSVAR Model (Eldo Level 74)		Yes
75 (PSP103)	PSP103 Model (Eldo Level 75)		Yes
101 (HVMOS)	HVMOS Model (Eldo Level 101)		Yes

Using -compat with MOSFET

Using the `-compat` runtime flag or selecting **`.OPTION COMPAT`** has the following effect on MOSFET Level values.

Table 4-33. MOS Levels with -compat

LEVEL	Model Name (Eldo Level)
2	Eldo2 (Eldo LEVEL 12)
3	Eldo3 (Eldo LEVEL 13)
6	Modified Lattin-Jenkins Grove Model (Eldo LEVEL 16)
8	Enhanced Berkeley LEVEL 2 (Eldo LEVEL 17)
13	Berkeley BSIM1 (Eldo LEVEL 8)
39	Berkeley BSIM2 (Eldo LEVEL 11)
49	Berkeley BSIM3 v3.0 & BSIM3 v3.1 (Eldo LEVEL 53)
50	Philips MOS Model 9 (Eldo LEVEL 59)
54	Berkeley BSIM4.0.0 (Eldo LEVEL 60)
57	Berkeley BSIMSOI3v2 PD (Eldo LEVEL 56, SOIMOD=1)
59	Berkeley BSIMSOI3v2 FD (Eldo LEVEL 56, SOIMOD=3)
68	HiSIM Model (Eldo LEVEL 66)
70	BSIMSOIv4 Model (Eldo LEVEL 72)

Computation of Effective Parasitic Areas and Perimeters

If using MOSFET without specifying parasitic areas and perimeters, please refer to "XA=VAL" on page 974 for more information about computation of effective parasitic areas and perimeters.

User Defined Models

By specifying `UDMP=1` on UDM models, Eldo will add the extrinsic contribution to the intrinsic part defined by the user. The parasitic effects are computed according to the Berkeley Level 1 common approach.

```
.model M NMOS level=21 UDMP=1
```

For more information on User Defined Models see the Introduction of the *Eldo UDM User's Manual*.

Output Quantities

For a listing of generic output quantities for MOSFET models see "MOSFET Plotting and Printing" on page 802. This lists the syntax used to plot or print the values of both extrinsic and intrinsic pins. The information is divided into subsections of output type and analysis type.

Noise in MOSFETs

Noise models are available for MOSFETs, see Noise Models in MOSFETs of the *Eldo Device Equations Manual*.

In Eldo, MOS noise models are selected using either the device model parameters **THMLEV** and **FLKLEV**, or the global Eldo options **THERMAL_NOISE**=VAL and **FLICKER_NOISE**=VAL.

Table 4-34. MOS Noise Models

THERMAL_NOISE/THMLEV Value	Noise Model
0	SPICE noise model—default[1]
1	Strong inversion model
2	Weak and strong inversion model
3	Previous model with short channel effects
4	Strong inversion model with short channel effects

1. Please note that the conductance used in the SPICE thermal noise equation is ***gm***. However, Eldo uses a more sophisticated computation of the total transistor conductance. This is given by the following equation:

$$gtot = gm + gds + gmb$$

Table 4-35. MOS Noise Models

FLICKER_NOISE/FLKLEV Value	Noise Model
0	SPICE noise model—default
1	Improved SPICE noise model
2	Improved SPICE noise model
3	Improved SPICE noise model

Selection of MOSFET Models via W/L Specifications (Binning)

Note

This functionality used to be set with option **MODWL** but has been the default since Eldo v5.1. Use option **NOMODWL** to switch off the functionality.

By default, MOS model versions are selected via **.MODEL** command **W** and **L** parameters (binning parameters). Whenever Eldo finds a MOS device for which the model name has no **.MODEL** command, it searches through all defined models for a model of the same root name and whose **W**/**L** range matches the specified device size. For example, in the case below, Eldo will assign the model `MODROOT.2` to the MOSFET `M1`:

```
M1 1 2 3 4 MODROOT w=2u l=3u
.MODEL MODROOT.1 NMOS VT0=1 WMIN=3u WMAX=5u LMIN=1u LMAX=5u
.MODEL MODROOT.2 NMOS VT0=2 WMIN=1u WMAX=5u LMIN=1u LMAX=5u
```

Note

Eldo selects the model to be assigned to MOS devices according to the geometric size of each device, even if these geometric sizes are modified at run-time via **.STEP** commands.

The separator in the **.MODEL** command should be a "." or a "_" character. If there are multiple occurrences of these characters in the model name, the last one specified is used as the separator. Specify option **MODWLDOT** to only allow "**.**" as the separator. When set, the character "_" can be used in the `<root>`. Examples for `<root><separator><extension>`:

```
Q1 vplus in out 0 QND_model area=10
.model QND_model_2 NPN AREAMIN=3 AREAMAX=20
```

Here, `<root> = QND_model`, `separator = _`, `<extension> = 2`.

```
Q1 vplus in out 0 QND.model area=10
.model QND.model_3 NPN AREAMIN=3 AREAMAX=20
```

Here, `<root> = QND.model`, `separator = _`, `<extension> = 3`.

The binning models algorithm works in the following way:

1. A model with dimensions W, L is chosen if the following formulae are satisfied:

$$WMIN + XWREF \times SCALM \leq \frac{W}{NF} \times SCALE + XW \times SCALM < WMAX + XWREF \times SCALM$$

$$LMIN + XLREF \times SCALM \leq LF \times SCALE + XL \times SCALM < LMAX + XLREF \times SCALM$$

Where ***XWREF***, ***XW***, ***XLREF***, ***XL***, ***SCALM*** are model parameters, ***SCALE*** is a global parameter set with **`.OPTION SCALE`**`=val`, and ***NF*** (number of fingers) is a MOS instance parameter found in BSIM4 and other recent models.

XL and ***XW*** default to 0.0. If ***XLREF*** is not specified, ***XLREF*** is set to ***XL***. If ***XWREF*** is not specified, ***XWREF*** is set to ***XW***. ***SCALM*** and ***SCALE*** default to 1.

2. If when applying this rule, no match is found, Eldo will repeat the search with two inclusive inequalities:

$$WMIN + XWREF \times SCALM \leq \frac{W}{NF} \times SCALE + XW \times SCALM \leq WMAX + XWREF \times SCALM$$

$$LMIN + XLREF \times SCALM \leq LF \times SCALE + XL \times SCALM \leq LMAX + XLREF \times SCALM$$

3. If a match is still not found, Eldo will issue an error.

Please also refer to “`MODWLDOT`” on page 988 and “`NOMODWL`” on page 989.

BSIM4 Model Selection

The BSIM4 model has unique conditions for W/L specification model selection. The conditions for this model are shown below:

$$WMIN \leq W / NF < WMAX \text{ and } LMIN \leq L < LMAX$$

Where ***NF*** is a parameter specified in the model instantiation card which defines the number of device fingers. An example is shown below:

```
.model nmos.1 nmos level=60 lmin=0.8u lmax=1.2u wmin=8u wmax=12u
.model nmos.2 nmos level=60 lmin=0.8u lmax=1.2u wmin=80u wmax=120u

m1 1 2 0 0 nmos w=10u nf=1 l=1u
m2 1 2 0 0 nmos w=100u nf=10 l=1u
```

In this case `m1` and `m2` will both use the `nmos.1` model.

MOS Parasitics Common Approach

The need arises to unify the handling of the parasitic elements for all Eldo models, which are under our management. That is to say, that proprietary models are not concerned by those changes. To summarize, Eldo Levels 1, 2, 3, 8 (BSIM1), 11 (BSIM2), 12, 13, 16, 17, 44 (EKV), 47 (BSIM3v2), 53 (BSIM3v3) and 59 (MOSP9) are relevant to this feature. This parasitics common approach deals with the following effects:

- Parasitics access resistance calculation.
- Drain, source area and perimeter calculations.
- Bulk diode current calculations.
- Bulk diode capacitance calculations.

Note

Noise calculations and geometric range definitions are also common.

Please refer to the Common Equations of the *Eldo Device Equations Manual*.

Four independent selectors **ALEV**, **RLEV**, **CAPLEV** and **DCAPLEV** ensure the switching between the different sets of equations. To ensure compatibility with previous versions of the software, an additional parameter **ARLEV** is necessary. **ARLEV** has priority over **ALEV** and **RLEV**, so if **ARLEV** is specified: **ALEV=RLEV=ARLEV**.

Table 4-36. MOSFET Common Parameters

Nr.	Name	Description	Default	Units
1	ACM	Flag to control which parasitic models are to be used	0	
2	OPTACM[1]	Flag to enable ACM control over parasitic models	0	
3	CALCACM[2]	Flag to control the area and perimeter calculations when ACM=12	0	
Parasitic Resistance Related Parameters				
4	RLEV	Switch selector	*	
5	RD	Drain ohmic resistance	0	Ω
6	RS	Source ohmic resistance	0	Ω
7	RSH	Drain and Source diffusion sheet resistance	0	Ω/sq
8	RDC	Additional drain resistance due to contact resistance	0	Ω
9	RSC	Additional source resistance due to contact resistance	0	Ω
10	LD	Lateral diffusion into channel from source and drain diffusion	0*	m
11	LDIF	Length of lightly doped diffusion adjacent to gate	0	m
12	HDIF	Length of heavily doped diffusion from contact to lightly doped region	0	m

Table 4-36. MOSFET Common Parameters

Nr.	Name	Description	Default	Units
13	LGCD	Gate to contact length of drain side	0	m
14	LGCS	Gate to contact length of source side	0	m
15	SC	Spacing between contacts	-1e20	m
16	RDD	Scalable drain resistance	0	Ω m
17	RSS	Scalable source resistance	0	Ω m
Area and Perimeter Related Parameters				
18	ALEV	Switch selector	*	
19	DL	Length account for masking and etching effects	0	
20	DW[3]	Width account for masking and etching effects	0	m
21	LMLT	Length diffusion layer shrink reduction factor	1	
22	WMLT	Width diffusion layer shrink reduction factor	1	
23	WD	Lateral diffusion into channel from bulk along width	0	
Bulk Diode Current Related Parameters				
24	DIOLEV	Switch selector	*	
25	JS	Bottom bulk junction saturation current density	0	Am^{-2}
26	JSW	Sidewall bulk junction saturation current density	0	Am^{-1}
27	IS	Bulk junction saturation current	1×10^{-14}	A
28	N	Emission coefficient	1	-
29	NDS	Reverse bias slope coefficient	1	-
30	VNDS	Reverse bias transition voltage	-1	V
31	SBTH	Flag for reverse diode behavior: 0 sets DIOLEV=1 and 1 sets DIOLEV=4	0	-
JUNCAP Diode Current Related Parameters (DIOLEV=9)				
32	VR	Voltage at which the parameters have been determined	0	V
33	JSGBR	Bottom saturation-current density due to electron-hole generation at $V = V_R$	1×10^{-3}	Am^{-2}
34	JSDBR	Bottom saturation-current density due to diffusion from back contact	1×10^{-3}	Am^{-2}
35	JSGSR	Sidewall saturation-current density due to electron-hole generation at $V = V_R$	1×10^{-3}	Am^{-1}

Table 4-36. MOSFET Common Parameters

Nr.	Name	Description	Default	Units
36	JSDSR	Sidewall saturation-current density due to diffusion from back contact	1×10^{-3}	Am^{-1}
37	JSGGR	Gate-edge saturation-current density due to electron-hole generation at $V = V_R$	1×10^{-3}	Am^{-1}
38	JSDGR	Gate-edge saturation-current density due to diffusion from back contact	1×10^{-3}	Am^{-1}
39	NBJ	Emission coefficient of the bottom forward current	1	
40	NSJ	Emission coefficient of the sidewall forward current	1	
41	NGJ	Emission coefficient of the gate-edge forward current	1	
42	CJBR	Bottom junction capacitance at $V = V_R$	1×10^{-12}	Fm^{-2}
43	CJSR	Sidewall junction capacitance at $V = V_R$	1×10^{-12}	Fm^{-1}
44	CJGR	Gate-edge junction capacitance at $V = V_R$	1×10^{-12}	Fm^{-1}
45	VDBR	Diffusion voltage of the bottom junction at $T = Tnom$	1	V
46	VDSR	Diffusion voltage of the sidewall junction at $T = Tnom$	1	V
47	VDGR	Diffusion voltage of the gate-edge junction at $T = Tnom$	1	V
48	PB	Bottom-junction grading coefficient	0.4	
49	PS	Sidewall junction grading coefficient	0.4	
50	PG	Gate-edge-junction grading coefficient	0.4	
51	TRDIO9	Nominal temperature for Juncap junction diode model	**TNOM** (**TR**)	°C
Bulk Diode Capacitance Related Parameters (DIOLEV≠9)				
52	DCAPLEV	Switch selector	*	
53	CBD	Zero bias Bulk-Drain capacitance	0	F
54	CBS	Zero bias Bulk-Source capacitance	0	F
55	CJ	Zero bias bottom Bulk junction capacitance	0*	Fm^{-2}
56	CJGATE	Zero bias gate-edge sidewall Bulk junction capacitance (only used when ALEV=3 and DCAPLEV=0)	0	Fm^{-1}
57	CJSW	Zero bias sidewall Bulk junction capacitance	0	Fm^{-1}
58	FC	Bulk junction forward bias capacitance coefficient	0.5*	
59	MJ	Bulk junction bottom grading coefficient	0.5	
60	MJSW	Bulk junction sidewall grading coefficient	0.33	
61	PB	Bulk bottom junction potential	0.8*	V

Table 4-36. MOSFET Common Parameters

Nr.	Name	Description	Default	Units
62	PBSW	Bulk sidewall junction potential	PB	V
63	TT	Transit time	0	s
Temperature Effect Related Model Parameters				
64	TNOM (TR)	Nominal temperature (TR for MOSP9 only)	27	°C
65	TMOD	Model temperature	TNOM	°C
66	TLEV	Temperature equation level selector	0	
67	TLEVC	Temperature equation level selector for capacitances and potentials	0	
68	CTA (TCJ)	Junction capacitance CJ temperature coefficient	0	$°K^{-1}$
69	CTP (TCJSW)	Junction sidewall capacitance CJSW temperature coefficient	0	$°K^{-1}$
70	PTA (TPB)	Junction potential PB temperature coefficient	0	$V°K^{-1}$
71	PTP (TPBSW)	Junction potential PHB temperature coefficient	0	$V°K^{-1}$
72	EG	Energy gap for PN junction diode	1.11[4]	eV
73	GAP1	First bandgap correction factor	7.02×10^{-4}	$eV°K^{-1}$
74	GAP2	Second bandgap correction factor	1108	°K
75	TLEVI (LIS)	Saturation current temperature selector	1	
76	ISTMP	Number of degrees that doubles IS value	10	°C
77	XTI	Saturation current temperature exponent	0	
78	TLEVR	Access resistances temperature selector	1	
79	TRD1 (TC1,TRD)	Temperature coefficient (linear) for RD	0	$°K^{-1}$
80	TRD2 (TC2)	Temperature coefficient (quadratic) for RD	0	$°K^{-2}$
81	TRS1 (TRS)	Temperature coefficient (linear) for RS	0	$°K^{-1}$
82	TRS2	Temperature coefficient (quadratic) for RS	0	$°K^{-2}$
83	TRSH1	Temperature coefficient (linear) for RSH	0	$°K^{-1}$
84	TRSH2	Temperature coefficient (quadratic) for RSH	0	$°K^{-2}$
Noise Effect Related Model Parameters				
85	AF	Flicker noise exponent	1	
86	KF	Flicker noise coefficient	0	
87	FLKLEV	Flicker noise level selector	0	

Table 4-36. MOSFET Common Parameters

Nr.	Name	Description	Default	Units
88	THMLEV	Thermal noise level selector	0	
Geometric Range Related Model Parameters				
89	WMIN	Model geometric range parameters. These model parameters give the range of the physical length and width dimensions to which the MOSFET model applies; used in conjunction with binning models	1	µm
90	WMAX		2000	µm
91	LMIN		1	µm
92	LMAX		100	µm

1. The model parameter OPTACM, when set to 1, has the same effect as .option ACM. The only difference is that the latter affects all of the model cards in the netlist, while the former affects the model card that it is specified in.
2. The model parameter CALCACM will accept either 0 (default) or 1. It is used only with .option ACM or OPTACM=1, and ACM=12. By default (CALCACM=0), sets the value of ALEV to 0 but if it is set to 1, it ALEV will be set to 2.
3. for **ALEV**=7, the definition is "total width connection."
4. if **TLEV**=0 or 1, **EG**=1.11, else **EG**=1.16
*****. For parameters marked with a ***** in their default parameter column, the default value depends upon the model selected, see tables below.

Table 4-37. RLEV Default Values

Model (Eldo Level)	RLEV default value
Level (12,13,16,17), BSIM1 (8)	0
EKV (44)	2
BSIM3v2 (47), BSIM3v3 (53)	4
Level (1,2,3), MOS9 (53), BSIM2 (11), MM11 (63,65)	6

Table 4-38. ALEV Default Values

Model (Eldo Level)	ALEV default value
Level (12,13,16,17), BSIM1 (8), BSIM3v3 (53)	0
EKV (44), BSIM3v2 (47)	2
Level (1,2,3), MOS9 (53), BSIM2 (11), MM11 (63,65)	6

Table 4-39. DIOLEV Default Values

Model (Eldo Level)	DIOLEV default value
Level (1,2,3), MOS9 (53), BSIM3v2 (47)	1
Level (12,13,16,17),	2
BSIM1 (8), BSIM2 (11)	3

Table 4-39. DIOLEV Default Values

Model (Eldo Level)	DIOLEV default value
EKV (44)	6
BSIM3v3 (53)	7
MM11 (63,65)	9

Table 4-40. DCAPLEV Default Values

Model (Eldo Level)	DCAPLEV default value
Level (12,13,16,17), BSIM1 (8)	0
BSIM3v2 (47)	1
Level (1,2,3), MOS9 (53), BSIM2 (11), EKV (44)	2
BSIM3v3 (53)	4
MM11 (63,65)	-

Scaling Rules

$$LDIF_{scal} = LDIF \cdot scalm \qquad HDIF_{scal} = HDIF \cdot scalm \cdot WMLT$$

$$DL_{scal} = DL \cdot scalm \qquad DW_{scal} = DW \cdot scalm$$

$$LD_{scal} = LD \cdot scalm \qquad WD_{scal} = WD \cdot scalm$$

$$AD_{scal} = \frac{AD}{scalm \cdot scalm} \qquad AS_{scal} = \frac{AS}{scalm \cdot scalm} vtvt$$

$$PD_{scal} = \frac{PD}{scalm} \qquad PS_{scal} = \frac{PS}{scalm}$$

$$JS_{scal} = \frac{JS}{scalm \cdot scalm} \qquad JSW_{scal} = \frac{JSW}{scalm}$$

$$CJ_{scal} = \frac{CJ}{scalm \cdot scalm} \qquad CJSW_{scal} = \frac{CJSW}{scalm}$$

$$CJGATE_{scal} = \frac{CJGATE}{scalm}$$

Note

The model parameter scaling factor *SCALM* can be defined for all model cards in the netlist using `.OPTION SCALM=val`, or can be individually defined for each model card using the model parameter `SCALM`. This overrides the global `SCALM` value defined using the `.OPTION` command.

Berkeley SPICE Models

The DC characteristics of the MOSFET are defined by the device parameters **VTO**, **KP**, **LAMBDA**, **PHI** and **GAMMA**. These are computed by Eldo, if the process parameters (**NSUB**, **TOX**, ...) are given. User specified values always override calculated values. **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**. Capacitances taken into account are the non-linear thin-oxide capacitance distributed among the gate, source, drain and bulk regions, and the non-linear 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**. The model is the piecewise linear voltage dependent capacitance model proposed by Meyer.

Some overlap among the parameters describing the junctions exists, for example Reverse current can be input either as **IS** (in Amperes) or as **JS** (in Am^{-2}). The first is an absolute value, while the second is multiplied by **AD** and **AS** to give the reverse current of the drain and source junctions respectively. This methodology is used as there is no sense in always relating junction characteristics with **AD** and **AS** of the device card; these areas can be defaulted. The same idea also applies to the zero-bias junction capacitances **CBD** and **CBS** (in Farads) on one hand, and **CJ** (in Fm^{-2}) on the other. Parasitic drain and source series resistance can be expressed as either **RD** and **RS** (in Ω) or **RSH** (in Ω/Square), the latter being multiplied by the number of squares **NRD** and **NRS** input on the device card.

The temperature of a semiconductor model can be defined in Eldo using the **TMOD** parameter. If this parameter is not present the global circuit temperature **TNOM** is assumed. The individual model temperature of a device can be set by using the optional instance parameter **T**.

Please refer to the MOS Level 1, 2, 3 Equations of the *Eldo Device Equations Manual*. For model parameters, please refer to the Berkeley SPICE Model (Eldo Level 1, 2, 3) Parameters of the *Eldo Device Equations Manual*.

Specific Levels 1, 2, and 3 initialization for parasitics common approach MOS parameters:

- **ALEV**=6; **RLEV**=6; **DIOLEV**=1; **DCAPLEV**=2
  if **XJ** is specified then **LD**=0.75×**XJ** otherwise **LD**=0;
- Equivalent name definitions:
  PHP parameter is equivalent to **PBSW**.
  XL and **XW** are equivalent to **DL** and **DW** respectively.

For details of the common parameters, see “MOS Parasitics Common Approach” on page 258.

MERCKEL MOSFET Models

Three Merckel model levels are presently available as Levels 4, 5 and 6. These models are defined by their own set of parameters, which can be easily extracted from measurements.

Note

Contact Mentor Graphics for more details about the above models.

Key Parameters

- **VT0**

  Threshold voltage at **VGS**=0.

- **MUO**

  Low field surface mobility.

- **D**

  Correction term issued from development of the '3/2' model. **D** is extracted from the model parameter.

- **TG**

  Gate field effect coefficient on mobility.

- **TD**

  Drain field effect coefficient on mobility, which equals $\frac{MUO}{(L \times VS)}$ where VS is the saturation velocity.

- **VEA**

  Early voltage coefficient.

The Level 4 model is a charge control model and the DC characteristics are the same as in Level 6.

The Level 5 model is very similar to that of SPICE Level 1, but it also takes into account the effect of mobility modulation by **TG**. The early effect is modeled by **KE** instead of **LAMBDA** as in SPICE models.

The Level 6 model is more accurate, and it is especially dedicated to submicron technologies; the threshold dependencies on geometries are modeled by **VE** instead of **LAMBDA** as in SPICE models.

The capacitance models are close to those of SPICE. The bulk junction capacitances vary with the power of −1/2 of the junction voltage. Overlap capacitances are computed from the **REC** parameter. **LDIF** is used for computation of the drain and source perimeters and areas, if the parameters for **AD**, **AS**, **PD** and **PS** are missing in the declaration of the MOS transistor.

Please refer to the MOS Level 4, 5, 6 Equations of the *Eldo Device Equations Manual*. For model parameters, please refer to the MERCKEL MOSFET Model (Eldo Level 4, 5, 6) Parameters of the *Eldo Device Equations Manual*.

For **FLKLEV** and **THMLEV**, see the “Noise in MOSFETs” on page 256.

Correspondence of Merckel to SPICE Model Parameters

For historical reasons, some SPICE parameter names have equivalent Merckel parameter names, which can be interchanged within a model definition. The following is a list of these parameters:

Table 4-41. Merckel-Spice

Merckel	SPICE
NIV	LEVEL
EOX	TOX
DPHIF	PHI
VTO	VT0
MUO	UO
KB	GAMMA
TG	THETA
NB	NSUB
CDIFS0	CJ
CDIFP0	CJSW
DL	2 LD
KO	KP/2

Berkeley SPICE BSIM1 Model (Eldo Level 8)

The Berkeley Short Channel IGFET Model (BSIM1) is a semi-empirical model proposed by the University of Berkeley (California). It was adopted to cope with rapid advances of technologies and fits well for submicron technologies.

The major effects modeled in BSIM1 include:

- mobility reduction due to the vertical field

- channel length modulation
- carrier velocity saturation
- non-uniform channel doping
- subthreshold conduction
- drain-induced barrier lowering
- source/drain charge sharing

A representation with 17 parameters per device size was found to be adequate for modeling the DC characteristics. Geometric dependencies are included to introduce parameter sensitivity to length and width. The charge model was derived from its drain-current counterpart to ensure model consistency of device physics. Charge conservation is also guaranteed. Also, three possible drain-source sharings are possible.

Please refer to the BSIM1 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Berkeley SPICE BSIM1 Model (Eldo Level 8) Parameters of the *Eldo Device Equations Manual*.

Specific BSIM1 initialization for parasitics common approach MOS parameters:

ALEV=0; **RLEV**=0; **DIOLEV**=3; **DCAPLEV**=0
if **XJ** is specified then **LD**=0.75×**XJ** otherwise **LD**=0;

For details of the common parameters, see “MOS Parasitics Common Approach” on page 258.

Berkeley SPICE BSIM2 Model (Eldo Level 11)

Based on BSIM1 model, the BSIM2 was developed by the University of Berkeley (California) to correct BSIM1 problems. BSIM2 has been successfully used to model the drain current and output resistance of MOSFETs with gate oxide as thin as 3.6 nm and channel length as small as 0.2 μm.

Based on recent physical understanding of deep-submicron MOSFETs, it has been found that the important effects that should be included in MOSFET modeling are:

- mobility reduction due to the vertical field
- carrier velocity saturation
- non-uniform channel doping
- subthreshold conduction

- drain-induced barrier lowering
- source/drain charge sharing
- channel length modulation
- source/drain parasitic resistance
- hot electron induced output resistance reduction
- inversion layer capacitance.

Except for the three last effects, most of these were already present in the BSIM1 model. In order to develop a consistent model, these effects were re-examined using new deep-submicron MOSFET considerations and their implementation. The charge model was derived from its drain-current counterpart to ensure model consistency of device physics. Charge conservation is also guaranteed. Also 3 possible drain-source sharings are possible.

Please refer to the BSIM2 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Berkeley SPICE BSIM2 Model (Eldo Level 11) Parameters of the *Eldo Device Equations Manual*.

Specific BSIM2 initialization for parasitics common approach MOS parameters:

ALEV=6; **RLEV**=0; **DIOLEV**=3; **DCAPLEV**=2
if **XJ** is specified then **LD**=0.75×**XJ**; otherwise **LD**=0;
CJ=0; **FC** is not a BSIM2 parameter, therefore it is set to 0.

For details of the common parameters, see the "MOS Parasitics Common Approach" on page 258.

Modified Berkeley Level 2 (Eldo Level 12)

A modified version of Level 2, Level 12 includes specification of Meyer capacitance parameters **CF1**, **CF2**, **CF3**, **CF5** and **CGBEX**. These parameters are used to solve the dynamic part of the model. The diode currents, impact ionization currents and access resistances are also calculated differently when using this model.

Please refer to the Modified Berkeley Level 2 (Eldo Level 12) of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Modified Berkeley SPICE Level 2 Parameters of the *Eldo Device Equations Manual*.

Modified Berkeley Level 3 (Eldo Level 13)

A modified version of Level 3, Level 13 includes specification of Meyer capacitance parameters **CF1**, **CF2**, **CF3**, **CF5** and **CGBEX**. These parameters are used to solve the dynamic part of the model. The diode currents, impact ionization currents and access resistances are also calculated differently when using this model.

> Please refer to the Modified Berkeley Level 3 (Eldo Level 13) of the *Eldo Device Equations Manual*.
> For model parameters, please refer to the Modified Berkeley SPICE Level 3 Parameters of the *Eldo Device Equations Manual*.

Example

```
*MOSFET model definition
.model me21 nmos level=4 vt0=1v eox=25n uo=600
+ nsub=2.0e16 phi=0.6 vmax=2.0e5 kw=2.24u
+ kl=2.24u gw=3.91u gl=0.7u dinf=0.1 kb=0.1
+ ve=1.0e4 ldif=10u cj=0.0001 cjsw=0 dw=0
+ dl=0.8u rec=0.15u tg=0.06
...
*main circuit
m1 vdd n3 n2 vbb me21 w=40u l=3u
```

Specifies the transistor m1 with a 40 micron channel width, a 3 micron channel length, drain connected to node vdd, gate to node n3, Source to node n2 and bulk to voltage vbb. The electrical parameters are specified in the model me21.

Modified Lattin-Jenkins-Grove Model (Eldo Level 16)

Level 16 model, also called Lattin-Jenkins-Grove model, is a model based on Level 2 equations and represents the ASPEC, ISPICE compatible model.

> Please refer to the Modified Lattin-Jenkins-Grove Model (Eldo Level 16) of the *Eldo Device Equations Manual*.
> For model parameters, please refer to the Modified Lattin-Jenkins-Grove Model Parameters of the *Eldo Device Equations Manual*.

Enhanced Berkeley Level 2 Model (Eldo Level 17)

An enhanced version of Berkeley Level 2, Level 17 includes modified equations for the description of the threshold voltage, subthreshold current, effective mobility and effective substrate doping.

The additional developments of the standard Berkeley SPICE Level 2 Model in this Eldo Level 17 are derived from model equations developed by the General Electric/Intersil Research Institutes.

Please refer to the Enhanced Berkeley Level 2 Model (Eldo Level 17) of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Enhanced Berkeley SPICE Level 2 Model Parameters of the *Eldo Device Equations Manual*.

EKV MOS Model (Eldo Level 44 or EKV)

The EKV model was originally developed by the EPFL (Ecole Polytechnique Fédérale de Lausanne), namely MM Enz, Krummenacher and Vittoz, hence the name. It is the result of many years of investigation to find a model that deals correctly with analog problems. Consequently, it may solve the problems of analog designers mainly in low voltage and low current applications and in terms of realistic behavior for currents, conductances and capacitances.

A compatible version of the EKV model is available in Eldo. This signifies that former versions of the model are also available. The compatibility covers versions v2.3 up to, and including, the new v2.6 (revision 2). The different versions are accessible through the model parameter **UPDATE** which can be used as shown in the table below. By default, the EKV model version v2.3 is selected.

Table 4-42. EKV MOS Models

Parameter Value	EKV Version
UPDATE=2.3 or UPDATE=23	EKV v2.3 (Default)
UPDATE=2.5 or UPDATE=25	EKV v2.5
UPDATE=2.61 or UPDATE=26.1	EKV v2.6 (Revision 1)
UPDATE=2.6 or UPDATE=26 UPDATE=2.62 or UPDATE=26.2	EKV v2.6 (Revision 2)
UPDATE=2.63 or UPDATE=26.3	EKV v2.6 (Revision 3)

Note

EKV versions v1.3 and v2.2 are no longer supported.

Please refer to the EKV MOSFET Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the EKV MOS Model (Eldo Level 44) Setup Parameters and EKV MOS Model (Eldo Level 44) Parameters of the *Eldo Device Equations Manual*.
Please refer to the Parameter units and Parameter preprocessing (intrinsic parameters initialization, version 2.6 revision 2) of the *Eldo Device Equations Manual*.

Berkeley SPICE BSIM3v2 Model (Eldo Level 47)

BSIM3 version 2 is a physical model developed by the University of Berkeley (California). It is based on a coherent quasi two-dimensional analysis of the MOSFET device structure, taking into account the effects of device geometry and process parameters. It allows users to accurately model MOSFET behavior for up-to-date submicron technologies.

Please refer to the BSIM3v2 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Berkeley SPICE BSIM3v2 Model (Eldo Level 47) Parameters of the *Eldo Device Equations Manual*.

Notes about Parameters

- **NPEAK**, **NGATE**, **ND** and **U0** may be entered in meters or centimeters:
  NPEAK is converted to cm^{-3} as follows: if **NPEAK** is greater than 1×10^{20}, it is multiplied by 1×10^{-6}.
  NGATE is converted to cm^{-3} as follows: if **NGATE** is greater than 1×10^{23}, it is multiplied by 1×10^{-6}.
  ND is converted to cm^{-3} as follows: if **ND** is greater than 1×10^{24}, it is multiplied by 1×10^{-6}.
  U0 is converted to $m^2(Vs)^{-1}$ as follows: if **U0** is greater than 1, it is multiplied by 1×10^{-4}.
  NSUB must be entered in cm^{-3}.
- Specific BSIM3v2 initialization for parasitics common approach MOS parameters:
  ARLEV=**ALEV**=2; **RLEV**=4; **DCAPLEV**=1; **DIOLEV**=1.
  CJ=5×10^{-4}; **CJSW**=5×10^{-10}; **PB**=1; **PBSW**=1;
  FC is not a BSIM3v2 parameter, therefore it is set to 0.
  As a consequence, you may use **LD**, **WD** instead of **DL**, **DW**. Respectively, if **DL**, **DW** are given, they will be used for common equation initializations instead of **LD** and **WD**.

For details of the common parameters, see the "MOS Parasitics Common Approach" on page 258.

- For derivative computation in BSIM3v2, a parameter **`DERIV`** has been added. By default, **`DERIV`**=1 for analytical derivatives. **`DERIV`** may be set to 0 (or with the command **`.option mnumer`**) for the finite difference method.

Berkeley SPICE BSIM3v3 Model (Eldo Level 53)

BSIM3v3 has been extensively modified from its previous release BSIM3v2. The physical effects modeled are the same as BSIM3v2. In addition to those, the new advancements are:

- A single drain current expression to describe current and output conductance characteristics from subthreshold to strong inversion as well as from the linear to the saturation operating region,
- New width dependencies for bulk charge and source/drain resistance (***Rds***),
- New capacitance models for both intrinsic and extrinsic capacitances,
- A new noise model,
- A new relaxation time model for characterizing the non-quasi-static effect for improved transient modeling.

Please refer to the BSIM3v3 Equations of the *Eldo Device Equations Manual*.

BSIM3v3 is available in four versions: BSIM3v3, BSIM3v3.1, BSIM3v3.2, and the most recent, BSIM3v3.3. According to Berkeley's recommendation, version BSIM3v3.2 is split into four versions.

All the different versions are accessible through the model parameter **`VER`**. By default, BSIM3v3.2.4 (**`VER`**=3.24) is selected.

Table 4-43. BSIM3v3 Models

Parameter Value	**BSIM3v3 Version**
VER(VERSION)=3.3	BSIM3v3.3
VER(VERSION)=3.24	BSIM3v3.2.4 (Default)
VER(VERSION)=3.23	BSIM3v3.2.3
VER(VERSION)=3.22	BSIM3v3.2.2 (Also if **`VER`** = 3.2)
VER(VERSION)=3.21	BSIM3v3.2.1
VER(VERSION)=3.1	BSIM3v3.1
VER(VERSION)=3.0	BSIM3v3

BSIM4 gate current inside BSIM3v3

Due to the continual decrease in device dimensions, the oxide thickness in new technologies is very small, so much so that the gate current is noticeable. Many companies would model the gate current with BSIM3v3 model in the form of a SUBCKT, however, this would slow the simulation a great deal. The gate current model of BSIM4 has, therefore, been incorporated inside BSIM3v3.

Having the gate current inside the compact model should be much faster than in the form of a SUBCKT. The model parameters used in BSIM4 should be used for the gate current. The two main selectors are ***IGCMOD*** & ***IGBMOD***. If either of these flags equal 1, the gate current will be calculated and the values taken will not be equal to 0. If both of these selectors are equal to 0 or not specified, the gate current is not calculated and the simulation will proceed as before.

These parameters can be found in the BSIM4 Gate Current Model parameter table.

BSIM4 Gate-Induced Drain and Source Leakage Current inside BSIM3v3

The gate induced drain leakage (GIDL) and gate induced source leakage (GISL) currents are gaining more importance in new technologies from day to day. Many companies would model the GIDL current with BSIM3v3 model, using a SUBCKT, however, this would slow the simulation. The GIDL (and GISL, see Note below) current model of BSIM4 has therefore been incorporated inside BSIM3v3 to increase the simulation speed. The model parameters used in BSIM4 should be used for the GIDL current. These four main parameters are ***AGIDL***, ***BGIDL***, ***CGIDL*** & ***EGIDL***. If any of these parameters are specified using the model command, the GIDL current will be calculated. If none of the parameters are specified however, the GIDL current is not calculated and the simulation will proceed as before.

These parameters can be found in the BSIM4 Gate-Induced Drain Leakage Model Parameters table.

Note

BSIM4.21 has the addition of Gate Induced Source Leakage (GISL) current. This is enabled by default in BSIM3v3 when enabling the GIDL current. If you want to use the GIDL model of BSIM4 versions prior to version 4.21, please set the ***IGIDLVER*** model parameter to a value less than 4.21 (default value). See the BSIM4.2.1 enhancements for further information on the GISL in BSIM4.

BSIMSOI3 DTOXCV Parameter inside BSIM3v3

There have been a number of problems regarding ***CAPMOD***=3 of BSIM3v3. The most serious symptom is the capacitance reduction (Cgg) in the strong inversion region if ***CAPMOD*** was changed from 0 to 3. BSIMPD is an SOI model formulated on top of the BSIM3v3 framework and also has the same trouble. The Berkeley team proposed an additional parameter, ***DTOXCV***, in ***CAPMOD***=3 of BSIMPDv2.2.3 to solve this issue. This has also been added to BSIM3v3.

This parameter can be found in the Process Parameters section of the parameter table.

Compatibility option for negative Rds values

By default, negative ***Rds*** values are clipped to zero. An Eldo option is available to allow negative ***Rds*** values: **.OPTION RDSWTPOS**=0|1. Default value is 1, which means Eldo clips negative ***Rds*** values (after temperature update) to zero. To allow negative ***Rds*** values, set the option to 0.

Caution

When this option is used to allow negative ***Rds*** values, the results may be unpredictable, or even cause the simulation to fail.

For model parameters, please refer to the Berkeley SPICE BSIM3v3 Model (Eldo Level 53) Parameters of the *Eldo Device Equations Manual*.

Notes about Parameters

- **NCH**, **NGATE** and **U0** may be entered in meters or centimeters:
  NCH is converted to cm^{-3} as follows: if **NCH** is greater than 1×10^{20}, it is multiplied by 1×10^{-6}.
  NGATE is converted to cm^{-3} as follows: if **NGATE** is greater than 1×10^{23}, it is multiplied by 1×10^{-6}.
  U0 is converted to m^2/Vsec as follows: if **U0** is greater than 1, it is multiplied by 1×10^{-4}.
  NSUB must be entered in cm^{-3}.
- Specific BSIM3v3 initialization for parasitics common approach MOS parameters:
  ARLEV=**ALEV**=0; **RLEV**=4; **DCAPLEV**=1; **DIOLEV**=1.
  CJ=5×10^{-4}; **CJSW**=5×10^{-10}; **PB**=1; **PBSW**=1;
  FC is not a BSIM3v3 parameter, therefore it is set to 0.
  As a consequence, you may use **LD**, **WD** instead of **LINT**, **WINT** or **DL**, **DW** instead of **DLC**, **DWC**. Respectively, **LINT**, **WINT** and **DLC**, **DWC** will be used for common equation initialization.

For details of the common parameters, see the "MOS Parasitics Common Approach" on page 258.

- As ***LINT*** and ***LD*** are both the same the user should only specify one parameter (***LINT*** or ***LD***). If both ***LINT*** and ***LD*** are specified then Eldo will return the following warning:

  ```
  Double definition for parameter(s) LD.
  ```

  Only the value of ***LD*** will be printed in the .chi file.

- For derivative computation in BSIM3v3, a parameter **`DERIV`** has been added. By default, **`DERIV`**=1 for analytical derivatives. **`DERIV`** may be set to 0 (or with the command **`.option mnumer`**) for the finite difference method.

- The different versions, BSIM3v3.2, BSIM3v3.1 and BSIM3v3, can also be selected by using the command **`.option bsim3ver`** = 3.2, 3.1 or 3.0.

- Parameter checking is added in BSIM3v3.2 to avoid bad values of certain parameters as follows:
  If ***PSCBE2*** ≤ 0.0, the user will be warned for the poor value used.
  If (***MOIN*** < 5.0) or (***MOIN*** > 25.0), a warning message will be given.
  If (***ACDE*** < 0.4) or (***ACDE*** > 1.6), a warning message will be given.
  If (***NOFF*** < 0.1) or (***NOFF*** > 4.0), a warning message will be given.
  If (***VOFFCV*** < -0.5) or (***VOFFCV*** > 0.5), a warning message will be given.
  If (***IJTH*** < 0.0), fatal error occurs.
  If (***TOXM*** <= 0.0), fatal error occurs

Printing and plotting BSIM3v3 Output States

The three capacitance states can be printed/plotted for any BSIM3v3 instance using the syntax `S(Mxx->state)`. For example, *CAPGDO* can be printed by specifying:

```
.PLOT DC S(Mxx->CAPGDO)      for DC or
.PLOT AC S(Mxx->CAPGDO)      for AC or
.PLOT TRAN S(Mxx->CAPGDO)    for TRAN
```

Table 4-44. Berkeley BSIM3v3 States

State	Description
CAPGDO	Gate-Drain overlap capacitance
CAPGSO	Gate-Source overlap capacitance
CAPGBO	Gate-Bulk overlap capacitance

Motorola SSIM Model (Eldo Level 54 or SSIM)

This model is only supported on Solaris platforms in Eldo 32-bit mode.

The Motorola SSIM model can be invoked in two ways:

- By specifying explicitly the level `SSIM` inside the **`.MODEL`** card:

  ```
  .MODEL mod NMOS LEVEL=SSIM <end of the model ...>
  ```

  or:

  ```
  .MODEL mod NMOS LEVEL=54 <end of the model ...>
  ```

- By running Eldo in Motorola mode:
  - by invoking Eldo with the `-ssim` switch at runtime, or
  - by adding **`.OPTION MOTOROLA`** at the beginning of the netlist (before the first **`.MODEL`** card).

When Eldo is in Motorola mode, the level of the SSIM model is 6:

```
.MODEL mod NMOS LEVEL=6
```

Note

This is useful when the user wants to utilize a **`.MODEL`** card from Motorola. The same system exists for the STMicroelectronics models.

Berkeley SPICE BSIMSOI3 v1.3 Model (Eldo Level 55)

BSIMSOI3 v1.3 is an officially released SOI (Silicon On Insulator) MOSFET model from the Device Group at the University of California at Berkeley. The model can be used for both Partially Depleted (PD) and Fully Depleted (FD) devices. Many advanced concepts are introduced so as to allow transition between PD and FD operation dynamically and continuously, namely the *Dynamic Depletion Approach*. The basic I-V model is modified from the BSIM3v3.1 equation set.

Please refer to the BSIMSOI3 v1.3 Equations of the *Eldo Device Equations Manual*.

Command Line Information

```
Mname <D node> <G node> <S node> <E node> [P node] <model>
+ [L=<VAL>] [W=<VAL>]
+ [AD=<VAL>] [AS=<VAL>] [PD=<VAL>] [PS=<VAL>]
+ [NRS=<VAL>] [NRD=<VAL>] [NRB=<VAL>]
+ [OFF] [BJTOFF=<val>] [RTH0=<VAL>] [CTH0=<VAL>]
+ [M=<VAL>] [DEBUG=<VAL>]
```

Parameters

- `D node`

  Drain node

- `G node`

  Gate node

- `S node`

  Source node

- `E node`

  Substrate node

- `[P node]`

  Optional external body contact
  if not specified, it is a 4-terminal device
  if specified, it is a 5-terminal device. The `P node` and `B node` will be connected by a resistance.

- `model`

  Level 55 BSIMSOI3 model name

- **L**=VAL

  Channel length

- **W**=VAL

  Channel width

- **AD**=VAL

  Drain diffusion area

- **AS**=VAL

  Source diffusion area

- **PD**=VAL

  Drain diffusion perimeter length

- **PS**=VAL

  Source diffusion perimeter length

- **NRS**=VAL

  Number of squares in source series resistance

- **NRD**=VAL

  Number of squares in drain series resistance

- **NRB**=VAL

  Number of squares in body series resistance

- **OFF**=VAL

  Device simulation off

- **BJTOFF**=VAL

  Turn off BJT current if equal to 1

- **RTH0**=VAL

  Thermal resistance per unit width
  if not specified, **RTH0** is extracted from the model card.
  if specified, it will override the one in the model card.

- **CTH0**=VAL

  Thermal capacitance per unit width
  if not specified, **CTH0** is extracted from model card.
  if specified, it will over-ride the one in model card.

- **M**=VAL

  This is the device "multiplier," simulating the effect of multiple devices in parallel. Effective width, overlap, junction capacitances, and junction currents are multiplied by **M**. Parasitic resistance values are divided by **M**. Default value is 1.

- **DEBUG**=VAL

  Please see the notes below.

Printing/Plotting States

The instance parameter **DEBUG** allows users to turn on debugging information selectively. Internal parameters (for example `par`) for an instance (for example `m1`) can be plotted by this command:

```
.plot <Analysis_Type> S(m1 -> par)
```

Example

```
.plot DC S(m1 -> body)
```

For model parameters, please refer to the Berkeley SPICE BSIMSOI3 v1.3 Model (Eldo Level 55) Parameters of the *Eldo Device Equations Manual*.

Berkeley SPICE BSIMSOI3 v2.x and v3.x Model (Eldo Level 56)

BSIMSOI3 is the officially released SOI (Silicon On Insulator) MOSFET model from the Device Group at the University of California at Berkeley. Both BSIMSOI v2.x and v3.x models can be used for Partially Depleted (**PD**) and Fully Depleted (**FD**) devices. For the BSIMSOIv2.x model many advanced concepts were introduced so as to allow transition between **PD** and **FD** operation dynamically and continuously, namely the *Dynamic Depletion Approach* (**DD**). The user is able to select one of these three modes using a parameter selector called **SOIMOD**. The basic I-V model is modified from the **BSIM3v3.1** equation set.

Please refer to the BSIMSOI3 v2.x and v3.x Equations of the *Eldo Device Equations Manual*.

The different versions are accessible through the model parameter **VERSION** as shown in the table below. By default, BSIMSOI3v3.2 (**VERSION**=3.2) is selected.

Table 4-45. BSIMSOI3 Version Selection

Parameter Value	BSIMSOI3 Version
VERSION=3.2	BSIMSOI3v3.2 (Default)
VERSION=3.11	BSIMSOI3v3.1.1
VERSION=3.1	BSIMSOI3v3.1
VERSION=3.0	BSIMSOI3v3.0
VERSION=2.23	BSIMSOI3v2.2.3
VERSION=2.22	BSIMSOI3v2.2.2
VERSION=2.21	BSIMSOI3v2.2.1
VERSION=2.1	BSIMSOI3v2.1

Parameter selector **SOIMOD** was an Eldo specific model parameter in the v2.x versions. Beginning version v3.0, it is a Berkeley standard model parameter to select between various SOI models: PD, FD and DD. For versions v3.0 and v3.1, the **SOIMOD** values have changed in Eldo to be compatible with the Berkeley standard values. Please see the following table:

Table 4-46. SOIMOD Selection

Model	SOIMOD for v2.x (Eldo specific)	SOIMOD for v3.0 (Spice compatible)	SOIMOD for v3.1 (Spice compatible)
PD	1	0 (default)	0 (default)
DD	2 (default)	-	-
FD	3	-	2
FD module over **PD**[1]	-	1	1

1. The FD module is an addition of some equations over the PD module to make the PD module also fit FD devices.

The different versions are handled separately inside this chapter. See "BSIMPDv2.x" on page 282, "BSIMFDv2.1" on page 282, "BSIMDDv2.1" on page 282, and "BSIMSOI3v3.x" on page 282.

The major features are summarized as follows:

- Dynamic depletion approach is applied on both I-V and C-V. Charge and Drain current are scalable with ***TBOX*** and ***TSI*** continuously.
- Supports external body bias and backgate bias; a total of 6 nodes.
- Real floating body simulation in both I-V and C-V. Body potential is properly bounded by diode and C-V formulation.
- Self heating implementation improved over the alpha version.
- An improved impact ionization current model.
- Various diode leakage components and parasitic bipolar current included.
- New depletion charge model (***EBCI***) introduced for better accuracy in capacitive coupling prediction. An improved BSIM3v3 based model is also included.
- Dynamic depletion can suit different requirements for SOI technologies.
- Single I-V expression as in BSIM3v3.1 to guarantee the continuity of I_{ds}, G_{ds} and G_m and their derivatives for all bias conditions.

TNODEOUT keyword

`TNODEOUT` is a keyword that can be specified in the instantiation statement of an SOI device as follows:

```
Mxx nd ng ns ne <np> <nb> <nT> mname <L=val> <W=val> TNODEOUT
```

- If **`TNODEOUT`** is not specified, the user can specify four nodes for a device with floating body. Specifying five nodes implies that the fifth node is the external body contact node, with a body resistance between the internal and external terminals. This configuration applies to a distributed body resistance simulation. Specifying six nodes implies a body contacted case with an accessible internal body node (sixth node). Specifying seven nodes implies that the seventh node is the temperature node. This may be used to model thermal coupling.
- If **`TNODEOUT`** is specified, simulation interprets the last node as the temperature node. You can specify five nodes for a device with floating body. Specifying six nodes implies body contact. Seven nodes is a body contacted case with an accessible internal body node.

VBSUSR keyword

VBSUSR is a keyword which allows you to set the transient initial condition of the body potential (V_b). For example:

```
.MODEL nnn NMOS LEVEL=56
m1 11 2 0 0 b nnn VBSUSR=1.5
```

BSIMSOI3v2.x

BSIMPDv2.x

BSIMPD is a Partially Depleted (**PD**) Silicon-on-Insulator (SOI) MOSFET model for SPICE simulation. This model is formulated on top of the BSIM3v3 framework. It shares the same basic equations with the bulk model so that the physical nature and smoothness of BSIM3v3 are retained. Most parameters related to general MOSFET operation (non-SOI specific) are directly imported from BSIM3v3 to ensure parameter compatibility.

Many enhanced features are included in BSIMPD through the joint effort of the BSIM Team at UC Berkeley and IBM Semiconductor Research and Development Center (SRDC) at East Fishkill. In particular, the model has been tested extensively within IBM on its state-of-the-art high speed SOI technology.

The latest version, BSIMPDv3.1.1, is implemented in Eldo. A version control parameter VERSION allows the use of older versions if required.

For model parameters, please refer to the Berkeley SPICE BSIMSOI3 PD Model (Eldo Level 56) Parameters of the *Eldo Device Equations Manual*.

BSIMFDv2.1

For model parameters, please refer to the Berkeley SPICE BSIMSOI3 FD v2.1 Model (Eldo Level 56) Parameters of the *Eldo Device Equations Manual*.

BSIMDDv2.1

For model parameters, please refer to the Berkeley SPICE BSIMSOI3 DD Model (Eldo Level 56) Parameters of the *Eldo Device Equations Manual*.

BSIMSOI3v3.x

Using BSIMPD as a foundation, a unified model is implemented for both PD and FD SOI circuit designs based on the concept of body-source built-in potential lowering.

In this version, BSIMSOI is constructed based on the concept of body-source built-in potential lowering, ΔV_{bi}. There are three modes (***soiMod*** = 0, 1, 2) in BSIMSOI: BSIMPD (***soiMod*** = 0) can be used to model the PD SOI device, where the body potential is independent of ΔV_{bi} (V_{BS} > ΔV_{bi}). Therefore the calculation of ΔV_{bi} is skipped in this mode. On the other hand, the ideal FD model (***soiMod*** = 2) is for the FD device with body potential equal to ΔV_{bi}. Hence the calculation of body current/charge, which is essential to the PD model, is skipped. For the unified SOI model (***soiMod*** = 1), however, both ΔV_{bi} and body current/charge are calculated to capture the floating-body behavior exhibited in FD devices.

Note
BSIMDD is not available for BSIMSOI3v3.x.

BSIMSOIv3.x Parameter List

For model parameters, please refer to the BSIMSOIv3.x Parameter List of the *Eldo Device Equations Manual*.

Philips MOS 9 Model (Eldo Level 59 or MOSP9)

Philips MOS 9 model is a compact model for MOS transistor, intended for the simulation of circuit behavior with emphasis on analog applications. This model has been developed originally by Philips Electronics, N.V and is now in the public domain.

Please refer to the MOS Model 9 Equations of the *Eldo Device Equations Manual*.

The implementation in Eldo is based on the unclassified report NL-UR 003/94 “MOS MODEL 9, level 902” issued in June 1995. The current implementation is MOS Model 9, level 903. In addition to the Philips syntax for the parameters, some equivalences to common approach parameters are made:

philips = **eldo**
LVAR = **DL**
LAP = **LD**
WVAR = **DW**
WOT = **WD**
NFR = **KF**
TR = **TNOM**

In these cases, the same default values are used.

Additionally, all the parameters available for the common approach are available for this model together with the corresponding equations set for parasitics. Furthermore, **AF** slope for noise has been added in Eldo. The overlap capacitances may be defined through the Philips parameter **COL** or through **CGDO**, **CGSO**. **CGBO** is also introduced.

The instantiation parameter **MULT** that indicates the number of devices in parallel is called **M** in Eldo.

The present restrictions in Eldo w.r.t. Philips implementation is that noise equations only include ***Sfl*** and ***Sth*** terms.

The model parameter **VERSION** can be set to values of 903.1 (default) or 903.2. When set to 903.2, it allows the model parameter **THE3R** to take negative values, otherwise **THE3R** is clipped to zero if negative.

Table 4-47. Philips MOS9 Version Selection

Parameter Value	Effect on THE3R
VERSION=903.1	Clip **THE3R** to zero if negative
VERSION=903.2	Allows **THE3R** to take negative values

For model parameters, please refer to the Philips MOS 9 Model (Eldo Level 59) Reference and Scaling Parameters of the *Eldo Device Equations Manual*.

Berkeley SPICE BSIM4 Model (Eldo Level 60)

The possible versions of this model are BSIM4.0.0, BSIM4.1.0, BSIM4.2.0, BSIM4.2.1, BSIM4.3.0, BSIM4.4.0, BSIM4.5.0, BSIM4.6.0, BSIM4.6.1, BSIM4.6.2, BSIM4.6.3, and BSIM4.6.4 (default).

Note

BSIM4 accepts the M factor.

According to the model developers, Berkeley, the BSIM4 model has been developed to explicitly address many issues in modeling sub-0.13 micron CMOS technology and RF high-speed CMOS circuit simulation.

Please refer to the BSIM4 Equations of the *Eldo Device Equations Manual*.

Use of Juncap diode (DIOLEV=9.0) with BSIM4

The Juncap diode (**DIOLEV**=9) can be used as a parasitic diode for the BSIM4 MOSFET model instead of that provided by Berkeley.

The Juncap diode can be chosen by specifying the model card parameter **DIOLEV**=9.0. Values for **DIOLEV** other than 9.0 will give a warning and the diode quantities will be calculated using the Berkeley parasitic diodes.

The parameters and equations of the Juncap diode (**DIOLEV**= 9.0) can all be found in the section Level 8 Equations of the *Eldo Device Equations Manual*.

Note

The initialization of device parameters (*PS*, *AS*, *PD*, *AD*) and calculation of geometrical quantities (*PSeff*, *ASeff*, *PDeff*, *ADeff*) are all Berkeley standard for BSIM4; not those of the common equations.

BSIM4 Model Selection via W/L Specifications

The BSIM4 model has unique conditions for W/L specification model selection. See "Selection of MOSFET Models via W/L Specifications (Binning)" on page 257.

The different BSIM4 versions are accessible through the model parameter VERSION. By default, BSIM4.6.4 (VERSION=4.64) is selected.

Table 4-48. Berkeley BSIM4 Version Selection

Parameter Value	BSIM4 Version
VERSION=4.64	BSIM4.6.4 (Default)
VERSION=4.63	BSIM4.6.3
VERSION=4.62	BSIM4.6.2
VERSION=4.61	BSIM4.6.1
VERSION=4.6	BSIM4.6.0
VERSION=4.5	BSIM4.5.0
VERSION=4.4	BSIM4.4.0
VERSION=4.3	BSIM4.3.0
VERSION=4.21	BSIM4.2.1
VERSION=4.2	BSIM4.2.0
VERSION=4.1	BSIM4.1.0
VERSION=4.0	BSIM4.0.0

BSIM4.x Model Parameters

- For model parameters, please refer to the Berkeley BSIM4 Model (Eldo Level 60) Parameters of the *Eldo Device Equations Manual*.
- For BSIM4.4.0 model parameters, please refer to the Berkeley BSIM4.4.0 Specific Model Parameters of the *Eldo Device Equations Manual*.
- For BSIM4.5.0 model parameters, please refer to the Berkeley BSIM4.5.0 Specific Model Parameters of the *Eldo Device Equations Manual*.
- For BSIM4.6.0 model parameters, please refer to the Berkeley BSIM4.6.0 Specific Model Parameters of the *Eldo Device Equations Manual*.

- For BSIM4.6.1 model parameters, please refer to the Berkeley BSIM4.6.1 Specific Model Parameters of the *Eldo Device Equations Manual*.
- For BSIM4.6.2 model parameters, please refer to the Berkeley BSIM4.6.2 Specific Model Parameters of the *Eldo Device Equations Manual*.

EKV3 MOS Model (Eldo Level 61 or EKV3)

The EKV model was originally developed by the EPFL (Ecole Polytechnique Fédérale de Lausanne), namely MM Enz, Krummenacher and Vittoz, hence the name. The EKV3 model is a MOSFET model that has five modes of operation, each mode covers the needs of certain cases.

The possible versions of this model are EKV301_01 and EKV301_02 (default).

Please refer to the EKV3 MOSFET Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the EKV3 MOS Model (Eldo Level 61) Parameters of the *Eldo Device Equations Manual*.

TFT Polysilicon Model (Eldo Level 62)

This is the modified polysilicon TFT model based on the original work at Rensselaer Polytechnic Institute (RPI).

This is a complete model developed for CAD and is based on the new universal charge control concept which guarantees stability and conversion. The unified DC model covers all regimes of operation and the AC model accurately reproduces frequency dispersion of capacitances (because of low mobility, carrier transit time is quiet; the signal period for even low frequency signals). This model provides automatic scaling of model parameters to accurately model a wide range of device geometries and physical based parameters can be easily extracted from experimental data.

Please refer to the TFT Polysilicon Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the TFT Polysilicon Model (Eldo Level 62) Parameters of the *Eldo Device Equations Manual*.

Philips MOS Model 11 Level 1101 (Eldo Level 63)

A new compact model for MOS transistors has been developed. Philips MOS Model 11 (MM11), the successor of Philips MOS Model 9, not only gives an accurate description of charges and currents and their first-order derivatives (transconductance, conductance, capacitances), but also of their higher-order derivatives. In other words it gives an accurate description of MOSFET distortion behavior, and as such MM11 is suitable for digital, analog as well as RF circuit design.

Please refer to the MOS Model 11 Level 1100 & 1101 Equations of the *Eldo Device Equations Manual*.

MOS Model 11 is a symmetrical, surface-potential-based model. It includes an accurate description of all physical effects important for modern and future CMOS technologies, such as, for example, gate tunnelling current, influence of pocket implants, poly-depletion, quantum-mechanical effects and bias-dependent overlap capacitances.

Level 1101 is an updated version of Level 1100. It uses the same basic equations as Level 1100, but uses different geometry scaling rules. It includes two types of geometrical scaling rules: physical rules and binning rules. Moreover, the temperature scaling has been implemented on the "miniset" level instead of the "maxiset" level as was the case for Level 1100.

The MM11 model (Level 1101) is implemented in Eldo as LEVEL=63.

For information on the MM11 model (Level 1100), see Philips MOS Model 11 Level 1100 (Eldo Level 65).

Binning is used with this MM11 Level 1101 to decide which geometrical scaling rule is used. For Physical rule **`BINNING`**=0.0 and for Binning rule **`BINNING`**=1.0. By Default **`BINNING`**=0.0 (Physical rule) is used.

Table 4-49. Philips MOS Model 11 Version Selection

Parameter Value	Geometrical scaling rule
BINNING=0.0	Physical rule (Default)
BINNING=1.0	Binning rule

Model Parameters

MM11 Level 1101(0) is specified with **`BINNING`**=0.0 (Physical rule).
For model parameters, please refer to the MM11 Level 1101(0) Model Parameters Physical Rule of the *Eldo Device Equations Manual*.

MM11 Level 1101(1) is specified with **`BINNING`**=1.0 (Binning rule).
For model parameters, please refer to the MM11 Level 1101(1) Model Parameters Binning Rule of the *Eldo Device Equations Manual*.

TFT Amorphous-Si Model (Eldo Level 64)

This is the modified amorphous-silicon TFT model based on the original work at Rensselaer Polytechnic Institute (RPI).

The model provides the following features and benefits:

- Uses the new, universal charge control concept, which guarantees stability and convergence
- Unified DC models cover all regimes of operation
- AC models accurately reproduces ***Cgc*** frequency dispersion
- Automatic scaling of model parameters to accurately model a wide range of device geometries
- Temperature dependence included
- A minimum number of physically based parameters that can easily be extracted from experimental data and related back to the fabrication steps

Please refer to the TFT Amorphous-Si Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the TFT Amorphous-Si Model (Eldo Level 64) Parameters of the *Eldo Device Equations Manual*.
For model parameters, please refer to the TFT Amorphous-Si Model Parasitic Parameters (Access Resistance and Overlap Capacitance Parameters) of the *Eldo Device Equations Manual*.

Printing or plotting a state from the TFT A-Si States structure

If a state from the list below is to be monitored by the user, the user has to type in the netlist for a given transistor M1 to monitor, for example Gm:

```
.PLOT DC S(M1->Gm) for DC or
.PLOT AC S(M1->Gm) for AC or
.PLOT TRAN S(M1->Gm) for TRAN
```

Philips MOS Model 11 Level 1100 (Eldo Level 65)

A new compact model for MOS transistors has been developed. Philips MOS Model 11 (MM11), the successor of Philips MOS Model 9, not only gives an accurate description of charges and currents and their first-order derivatives (transconductance, conductance, capacitances), but also of their higher-order derivatives. In other words it gives an accurate description of MOSFET distortion behavior, and as such MM11 is suitable for digital, analog as well as RF circuit design.

MOS Model 11 is a symmetrical, surface-potential-based model. It includes an accurate description of all physical effects important for modern and future CMOS technologies, such as, for example, gate tunnelling current, influence of pocket implants, poly-depletion, quantum-mechanical effects and bias-dependent overlap capacitances.

There are two versions of this model. Level 1100 and Level 1101. Level 1101 is an updated version of Level 1100.

The MM11 model (Level 1100) is implemented in Eldo as LEVEL=65.

For information on the MM11 model (Level 1101), see Philips MOS Model 11 Level 1101 (Eldo Level 63).

Please refer to the MOS Model 11 Level 1100 & 1101 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the MM11 Level 1100 Model (Eldo Level 65) Parameters of the *Eldo Device Equations Manual*.

HiSIM Model (Eldo Level 66)

According to the model developers, HiSIM (**Hi**roshima University **S**TARC **I**GFET **M**odel) is the first complete surface-potential-based MOSFET model for circuit simulation based on the drift-diffusion approximation.

The most important advantage of the surface-potential-based modeling is the unified description of device characteristics for all bias conditions. The physical reliability of the drift-diffusion approximation has been proved by 2D device simulations with channel lengths even down to below 0.1µm. To obtain analytical solutions for describing device performances, the charge sheet approximation of the inversion layer with zero thickness has been introduced. Together with the gradual-channel approximation all device characteristics are then described analytically by the channel-surface potentials at the source side (ϕ_{S0}) and at the drain side (ϕ_{SL}). These surface potentials are functions of applied voltages on the four MOSFET terminals; the gate voltage V_g, the drain voltage V_d, the bulk voltage V_b and the reference potential of the source V_s. This is the long-channel basis of the HiSIM model, and extensions of the model approximations are done for advanced technologies. All newly appearing phenomena such as short-channel and reverse-short-channel effects are included in the surface potential calculations causing modifications resulting from the features of these advanced technologies.

HiSIM versions are accessible through the model parameter VERSION. By default, HiSIM2.4.3 (VERSION=243) is selected, Table 4-50 shows how to select the other available versions.

Table 4-50. HiSIM Version Selection

Parameter Value	HiSIM Version
VERSION=243	HiSIM2.4.3 (Default)
VERSION=242	HiSIM2.4.2
VERSION=241	HiSIM2.4.1
VERSION=240	HiSIM2.4.0

Table 4-50. HiSIM Version Selection

Parameter Value	HiSIM Version
VERSION=231	HiSIM2.3.1

Please refer to the HiSIM Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the HiSIM (Eldo Level 66) Model Parameters of the *Eldo Device Equations Manual*.

SP Model (Eldo Level 67)

SP is a generic compact MOSFET model developed at The Pennsylvania State University. It is surface-potential-based, free from unphysical behavior often associated with more traditional models and contains a relatively small number of parameters. The development of SP is based on solution of several long standing problems of compact MOSFET modeling.

Consequently SP is a surface potential based model that does not contain iterative loops or channel segmentation in both the intrinsic and the extrinsic submodels.

Please refer to the Surface-Potential-Based Compact MOSFET Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Ranges of SP Parameters, Temperature dependence (-55 to 150), and Extrinsic Model Parameters of the *Eldo Device Equations Manual*.

Philips MOS Model 11 Level 1102 (Eldo Level 69)

MOS Model 11 (MM11, level 1102) is a new compact MOSFET model, intended for digital, analog and RF circuit simulation in modern and future CMOS technologies. MM1102 gives not only an accurate description of currents and charges and their first-order derivatives (i.e. transconductance, conductance, capacitances), but also of the higher order derivatives, resulting in an accurate description of electrical distortion behavior. The latter is especially important for analog and RF circuit design. The model furthermore gives an accurate description of the noise behavior of MOSFETs. Additionally, in order for the model to be valid for modern and future MOS devices, several important physical effects have been included in the model.

MM1102, is an updated version of MM1101. It uses slightly different equations. The surface potential generally is implicitly related to the terminal voltages and has to be calculated iteratively. Since the iterative procedure was assumed to be time consuming, the surface potential has been approximated by an explicit expression in MM1101. In the MM1102, the surface potential is calculated iteratively using a second-order Newton-Raphson procedure, resulting in a much more accurate description of surface potential which is obtained within three iterations. Owing to the increased accuracy, some of the basic equations used in MM1101 can be simplified, and as a result MM1102 is computationally as fast as MM1101. In addition, a more physical and simpler velocity saturation expression is used, and as a consequence the saturation voltage expression has changed slightly as well. This all results in a more accurate description of transconductance in saturation.

The MM11 model (Level 1102) is implemented in Eldo as LEVEL=69.

Please refer to the MOS Model 11 Level 1102 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the MM11 Level 1102 Model Parameters (Electrical), MM11 Level 11020 Model Parameters (Physical), and MM11 Level 11021 Model Parameters (Binning) of the *Eldo Device Equations Manual*.

Philips PSP Model (Eldo Level 70)

The PSP model is a new compact MOSFET model, which has been jointly developed by Philips Research and Penn State University (currently under development in Arizona State University). It is a surface-potential based MOS Model, containing all relevant physical effects (mobility reduction, velocity saturation, DIBL, gate current, lateral doping gradient effects, and so on) to model present-day and upcoming deep-submicron CMOS technologies. Unlike previous Philips MOS models, the source/drain junction model, c.q. the JUNCAP2 model, is an integrated part of the PSP model.

The PSP model is a symmetrical, surface-potential-based model, giving an accurate physical description of the transition from weak to strong inversion. The PSP model includes an accurate description of all physical effects important for modern and future CMOS technologies

In addition, it gives an accurate description of charges and currents and their first-order derivatives (transconductance, conductance, capacitances), but also of their higher-order derivatives. In other words, it gives an accurate description of MOSFET distortion behavior, and as such the PSP model is suitable for digital, analog as well as RF circuit design.

PSP model versions are accessible through the VERSION model parameter. The possible values of VERSION are listed in Table 4-54.

Table 4-51. PSP Version Selection

Parameter Value	PSP Version
VERSION=102.33	PSP 102.3.3 (Default)
VERSION=102.32	PSP 102.3.2
VERSION=102.3	PSP 102.3
VERSION=102.21	PSP 102.2.1
VERSION=102.2	PSP 102.2

Table 4-51. PSP Version Selection

Parameter Value	PSP Version
VERSION=102.1	PSP 102.1
VERSION=102.0	PSP 102.0

Please refer to the PSP Model Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Parameter Scaling and Parameter Sets of the *Eldo Device Equations Manual*.

Philips MOS Model 20 Level 2002 (Eldo Level 71)

According to the model developers, Philips, MOS Model 20 (MM20) is a new compact MOSFET model, intended for analog circuit simulation in high-voltage MOS technologies. MOS Model 20 describes the electrical behavior of the region under the thin gate oxide of a high-voltage MOS device, like a Lateral Double-diffused MOS (LDMOS) device or an extended-drain MOSFET. It thus combines the MOSFET-operation of the channel region with that of the drift region under the thin gate oxide in a high-voltage MOS device. As such, MOS Model 20 is aimed as a successor of the combination of MOS Model 9 (MM9) for the channel region in series with MOS Model 31 (MM31) for the drift region under the thin gate oxide, in macro models of various high-voltage MOS devices.

Note

The MM20 level 2001 model is not supported.

The MOS Model 20 (MM20, level 2002.2) is implemented in Eldo as LEVEL=71.

Please refer to the MOS Model 20 Level 2002 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the MM20 Level 2002 Geometrical Model Parameters (Eldo Level 71) of the *Eldo Device Equations Manual*.

Berkeley SPICE BSIMSOI4.0 Model (Eldo Level 72)

BSIMSOI is a SPICE compact model for SOI (Silicon-On-Insulator) circuit design. According to the model developers, Berkeley, the BSIMSOI4.0 model is formulated 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. BSIMSOI4.0 addresses several new issues in modeling sub-0.13 micron CMOS/SOI high-speed and RF circuit simulation. BSIMSOI4.0 is fully backward compatible with its previous 3.x version.

The BSIMSOI4.0 model is implemented in Eldo as LEVEL=72.

BSIMSOI4 model versions are accessible through the VERSION model parameter. The possible values of VERSION are listed in Table 4-52.

Table 4-52. BSIMSOI4 Version Selection

Parameter Value	BSIMSOI4 Version
Version=4.2	BSIMSOI4.2 (Default)
Version=4.0	BSIMSOI4.0

Please refer to the BSIMSOI4 Equations of the *Eldo Device Equations Manual*.
For model parameters, please refer to the Berkeley BSIMSOI4 Model (Eldo Level 72) Parameters of the *Eldo Device Equations Manual*.

HiSIM-LDMOS Model (Eldo Level 73)

HiSIM-LDMOS has been developed as an extension of HiSIM for conventional MOSFETs. According to the model developers, HiSIM (**Hi**roshima University **S**TARC **I**GFET **M**odel) is the first complete surface-potential-based MOSFET model for circuit simulation based on the drift-diffusion approximation.

HiSIM-LDMOS shares some significant equations with the HiSIM Model but has differences due to the drift region and self-heating. For information on HiSIM, see “HiSIM Model (Eldo Level 66)” on page 289.

The most important advantage of the surface-potential-based modeling is the unified description of device characteristics for all bias conditions. The physical reliability of the drift-diffusion approximation has been proved by 2D device simulations with channel lengths even down to below 0.1µm. To obtain analytical solutions for describing device performances, the charge sheet approximation of the inversion layer with zero thickness has been introduced. Together with the gradual-channel approximation all device characteristics are then described analytically.

The most important feature of LDMOS, different from the conventional MOSFET, is originated by the drift region introduced to achieve high voltage applications. By varying the length as well as a concentration of the drift region, various devices with various operating bias conditions are realized.

The HiSIM-LDMOS model is implemented in Eldo as LEVEL=73.

HiSIM-LDMOS versions are accessible through the VERSION model parameter. By default, HiSIM-LDMOS/HV 1.1.1 (VERSION=111) is selected. Table 4-53 shows the model parameter values for the available versions.

Table 4-53. HiSIM-LDMOS Version Selection

Parameter Value	HiSIM-LDMOS/HV Version
VERSION=111 or 1.11	HiSIM-LDMOS/HV 1.1.1 (Default)
VERSION=110 or 1.10	HiSIM-LDMOS/HV 1.1.0
VERSION=102 or 1.02	HiSIM-LDMOS/HV 1.0.2
VERSION=101 or 1.01	HiSIM-LDMOS/HV 1.0.1
VERSION=100 or 1.00	HiSIM-LDMOS/HV 1.0.0-SC3

Please refer to the HiSIM-LDMOS Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the HiSIM-LDMOS (Eldo Level 73) Model Parameters of the *Eldo Device Equations Manual*.

MOSVAR Model (Eldo Level 74)

According to the model developers, the MOS varactor compact model is based in part on the PSP MOSFET model and is intended for analogue and RF design. It includes dynamic inversion, finite poly doping, quantum mechanics, tunneling currents, and parasitics to model advanced MOS technologies.

The MOSVAR device is instantiated in Eldo as a 3-terminal MOSFET.

The MOSVAR model has been implemented in Eldo as LEVEL=74.

MOSVAR model versions are accessible through the VERSION model parameter. The possible values of VERSION are listed in Table 4-54.

Table 4-54. MOSVAR Version Selection

Parameter Value	MOSVAR Version
VERSION=1.1	MOSVAR 1.1 (Default)
VERSION=1.0	MOSVAR 1.0

Please refer to the MOSVAR Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the MOSVAR (Eldo Level 74) Model Parameters of the *Eldo Device Equations Manual*.

PSP103 Model (Eldo Level 75)

There are two versions of this model; PSP 103.0 and PSP 103.1 (the default).

According to the joint-developers, NXP Semiconductors Research (formerly part of Philips) and Arizona State University (formerly at The Pennsylvania State University), the PSP (Version 103.0) model is a new compact MOSFET model. The roots of PSP lie in both MOS Model 11 (Philips) and SP (Penn State). PSP is a surface-potential based MOS Model, containing all relevant physical effects (mobility reduction, velocity saturation, DIBL, gate current, lateral doping gradient effects, STI stress, etc.) to model present-day and upcoming deep-submicron bulk CMOS technologies. The source/drain junction model (the JUNCAP2 model) is fully integrated in PSP.

PSP not only gives an accurate description of currents, charges, and their first order derivatives (i.e. transconductance, conductance and capacitances), but also of the higher order derivatives, resulting in an accurate description of electrical distortion behavior. The latter is especially important for analog and RF circuit design. The model furthermore gives an accurate description of the noise behavior of MOSFETs. Finally, PSP has an option for simulation of non-quasi-static (NQS) effects.

Please refer to the PSP103 Model Equations of the *Eldo Device Equations Manual*. For model parameters, please refer to the Parameter Scaling and Parameter Sets of the *Eldo Device Equations Manual*.

HVMOS Model (Eldo Level 101)

The HV (High-Voltage) MOS transistor model is based on the BSIM3v3 model. Major enhancements include current-crowding effect at high gate bias, asymmetric source-drain structure, self-heating, and more flexible gate-dependent output characteristics. Like BSIM3v3, the HVMOS transistor model also allows the binning option to achieve even higher accuracy. The binning equation is given by:

$$P = P0 + \frac{Pl}{Leff} + \frac{Pw}{Weff} + \frac{Pp}{Leff \cdot Weff}$$

Note

Eldo defaults ***LMIN*** and ***WMIN*** to 1×10^{-6} m if the user did not specify these parameters. To avoid mistakes in finding the correct models, please set the ***LMIN*** and ***WMIN*** values in your model cards.

HVMOS License

The HVMOS model is a proprietary model of Cadence Design Systems, Inc. To use the HVMOS model, a license for Eldo from Mentor Graphics is required, *as well as* a license for the HVMOS model from Cadence.

To setup the license daemon for the HVMOS library, please refer to BTA's "License Installation and Management User Guide."

Please refer to the HVMOS Model of the *Eldo Device Equations Manual*.
For model parameters, please refer to the HVMOS Model (Eldo Level 101) Parameters of the *Eldo Device Equations Manual*.

S-Domain Filter

```
FNSxx IN OUT [RIN=val] [ROUT=val] NN {NN}, DN {DN}
```

The Eldo S-Domain filter is defined by the Transfer Function:

$$H(s) = \frac{N_0 + N_1 s + N_2 s^2 + N_3 s^3 + \ldots + N_n s^n}{D_0 + D_1 s + D_2 s^2 + D_3 s^3 + \ldots + D_m s^m}$$

Parameters

- `xx`

  S-Domain filter name.
- `IN`

  Name of the input node.
- `OUT`

  Name of the output node.
- **`RIN`**

  Input resistance. Default is infinity. This resistance of value **`RIN`** is located between node `IN` and ground. Note that setting **`RIN`** to 0 will have the default effect, i.e. that **`RIN`** will be infinity.
- **`ROUT`**

  Output resistance. Default is 0.0. FNS output is equivalent to a voltage source in series with an output resistance **`ROUT`**.
- `NN`

  Coefficients of the Transfer Function numerator, starting from N_0.
- `DN`

  Coefficients of the Transfer Function denominator, starting from D_0.

Example

```
fns1 n1 n2 1 2, 1 3 2
```

Specifies an S-Domain filter fns1 with input and output nodes n1 and n2. The Transfer Function is:

$$H(s) = \frac{1 + 2s}{1 + 3s + 2s^2}$$

It is possible to specify the order of the coefficient in brackets after giving the coefficient value. All non-specified coefficients will be set to 0, for example:

```
fns2 1 2 rout=1 1.0e0(0), 1.0e0(0) 3.0e-8(1) 1.0e-16(3)
```

is equivalent to:

```
fns2 1 2 rout=1 1.0e0, 1.0e0 3.0e-8 0 1.0e-16
```

Z-Domain Filter

```
FNZxx IN OUT FREQ=VAL [RIN=val] [ROUT=val] NN {NN}, DN {DN}
```

The Eldo Z-Domain filter is defined by the Transfer Function:

$$H(z) = \frac{N_0 + N_1 z^{-1} + N_2 z^{-2} + N_3 z^{-3} + \ldots + N_n z^{-n}}{D_0 + D_1 z^{-1} + D_2 z^{-2} + D_3 z^{-3} + \ldots + D_m z^{-m}}$$

Parameters

- `xx`

  Z-Domain filter name.

- `IN`

  Name of the input node.

- `OUT`

  Name of the output node.

- **FREQ**=VAL

  Sampling frequency in Hertz.

- **RIN**

  Input resistance. Default is infinity. This resistance of value **RIN** is located between node `IN` and ground. Note that setting **RIN** to 0 will have the default effect, i.e. that **RIN** will be infinity.

- **ROUT**

  Output resistance. Default is 0.0. FNZ output is equivalent to a voltage source in series with an output resistance **ROUT**.

- `NN`

  Coefficients of the Transfer Function numerator, starting from N_0.

- `DN`

  Coefficients of the Transfer Function denominator, starting from D_0.

The AC response of Z transforms was modified, beginning Eldo v6.3, to be consistent with that of ADMS. The term *sinx/x* is taken into account by Eldo. Use the option **NOZSINXX** to remove the effect of that term for pre-v6.3 functionality.

Example

```
fnz1 n1 n2 freq=1k 1 3, 1 2 4
```

A Z-Domain filter fnz1 with input and output nodes n1 and n2, clocked at 1kHz. The Transfer Function is:

$$H(z) = \frac{1 + 3z^{-1}}{1 + 2z^{-1} + 4z^{-2}}$$

It is possible to specify the order of the coefficient in brackets after giving the coefficient value. All non-specified coefficients will be set to 0, for example:

```
fnz2 1 2 freq=1k rout=1 1.0e0(0), 1.0e0(0) 3.0e-8(1) 1.0e-16(3)
```

is equivalent to:

```
fnz2 1 2 freq=1k rout=1 1.0e0, 1.0e0 3.0e-8 0 1.0e-16
```

Subcircuit Instance

```
Xxx NN {NN} NAME [PAR=VAL] [PAR={EXPR}] [M=VAL] [TEMP=VAL] [DTEMP=VAL]
+ [STATISTICAL=0|1] [(ANALOG|OSR|DIGITAL)] [NONOISE|NOISE=0]

Xxx [MODEL:] MNAME PIN: {pin=net}
+ PARAM: {par=val} KEYWORD: {keywords} [STATISTICAL=0|1]

Xxx [MODEL:] MNAME NET: {net=pin}
+ PARAM: {par=val} KEYWORD: {keywords} [STATISTICAL=0|1]
```

This statement is used to instantiate a subcircuit that has been previously defined using a **.SUBCKT** command. Subcircuit definitions and instances can be nested. Parameters contained in a subcircuit can be assigned explicitly or via the **.PARAM** command, direct assignment always taking precedence over **.PARAM** commands.

The first syntax above shows pins mapped by position. The second syntax above shows pins mapped by name. Usually, pins of subcircuit (X) instances are supposed to be specified in the order corresponding to that of the **.SUBCKT** declaration (first syntax). However, it is also possible to instantiate a subcircuit (X) using pin names of a **.SUBCKT** definition (second syntax), providing that option **XBYNAME** is set. The third syntax is similar to the second, with nets mapped to pins.

In order to improve execution speed the subcircuit can be optionally solved using the differentiated accuracy system.

Before using this system, refer to "Speed and Accuracy" on page 1061.

Parameters

- `xx`

  Subcircuit name.

- `NN`

  Names of the nodes to be connected externally. Nodes are referenced in the order they appear in a **.SUBCKT** command.

- `NAME`

  Name of the subcircuit being instantiated, as specified by the **.SUBCKT** command.

- `PAR=VAL`

  Specifies that the parameter `PAR` is assigned the value `VAL` inside the subcircuit. This parameter assignment takes precedence over any parameter assignments occurring in the **.SUBCKT** command.

- **M**=VAL

  Multiplication factor for the subcircuit instantiation. This effectively places **M** instances of the subcircuit in parallel, all connected to the specified nodes NN. If **M**=0 is specified, the corresponding subcircuit instance will be ignored (as if the subcircuit instance is commented out). Every element defined in the subcircuit will be duplicated by this multiplication factor.

- **TEMP**=VAL

  Sets temperature for the individual subcircuit. This overrides the temperature of devices which are instantiated in the X instance. Default nominal temperature=27 °C.

- **DTEMP**=VAL

  Temperature difference between the devices in the subcircuit and the rest of the circuit, in degrees Celsius. Default value is 0.0.

Note

TEMP and **DTEMP** are mutually exclusive. If both are specified, the last one is used.

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the subcircuit instance. 0 means the subcircuit components will keep their nominal values. 1 means the subcircuit components have statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- **(ANALOG)**

  Keyword used with the differentiated accuracy system indicating that the subcircuit should be solved using Newton block iteration techniques. These techniques are used in conjunction with the **EPS** parameter in the **.OPTION** command. **(ANALOG)** basically means “high accuracy.”

Before using the differentiated accuracy system see the relevant section in “Speed and Accuracy” on page 1061.

- **(OSR|DIGITAL)**

  Used to stop propagation of the **ANALOG** flag across the hierarchy. In addition the flag will request Eldo to use OSR in the selected blocks, if possible (i.e. MOS subcircuit with **OSR** flag could then be solved by OSR, but BJT subcircuit will still be solved by Newton even if flag **OSR** is set).

- **NONOISE**

  Specifies that no noise model will be used for this subcircuit when performing noise analysis. Therefore, the subcircuit presents no noise contribution to the noise analysis. For more details see “.SUBCKT” on page 898. Can also be specified between parentheses as **(NONOISE)** or with **NOISE**=0.

- **NOISE**=0

  Synonymous with **NONOISE** keyword. Specifies that no noise model will be used for this subcircuit when performing noise analysis. Therefore, the subcircuit presents no noise contribution to the noise analysis. Specifying any value other than zero will cause Eldo to issue a warning, and the parameter will be ignored.

- [**MODEL:**] MNAME

  Name of the subcircuit being instantiated, as specified by the **.SUBCKT** command.

- **PIN:** {pin=net} | **NET:** {net=pin}

  A list of pins to be mapped by name. pin is the name of the pin as declared in a **.SUBCKT** command. net is the name of the node to be connected externally ("parent" name). Eldo also accepts **NET:** in place of **PIN:**, in which case the order is reversed, Eldo will first expect the actual net named followed by the pin name as it appears in the **.SUBCKT** command.

- **PARAM:** {par=val}

  Specifies that the parameter PAR is assigned the value VAL inside the subcircuit. This parameter assignment takes precedence over any parameter assignments occurring in the **.SUBCKT** command.

- **KEYWORD:** {keywords}

  Allows specification of a list of keywords: **SWITCH**, **ANALOG**, **OSR**, **DIGITAL**.

Notes for pins mapped by name syntax

- When mapping pins by name, keywords **PIN:**, **PARAM:**, and **KEYWORD:** are mandatory in order to separate the different fields.
- Fields cannot be split, the same keyword cannot appear twice in the same X instance.
- Fields can be specified in any order.
- Eldo will issue an error if some mappings are missing in the **PIN:** or **NET:** declarations.
- When option **XBYNAME** is set, it is possible to mix both syntaxes in the same netlist, Eldo will check for the presence of at least one of the keywords **PIN:**, **PARAM:**, **NET:**, **MODEL:**, or **KEYWORD:** to select which syntax to be used for each X instance.
  See option "XBYNAME" on page 962.

Combining Identical Subcircuits

Multiple identical subcircuit instances connected in parallel are reduced into a single instance using the **M** parameter, for example:

```
X1 1 2 FOO A=1 B=1
X2 1 2 FOO A=1 B=1
```

```
X3 1 2 FOO A=1.0 B=1
```

Here, `X3` will remain as it is because the character string for `A` does not match, but X instances `X1` and `X2` will be replaced by:

```
X1 1 2 FOO A=1 B=1 M=2
```

For more information see "Merging Devices in Parallel" on page 120.

Examples

```
*SUBCKT definition
.subckt inv 1 2
r1 1 3 2k
r2 3 4 4k
r3 4 2 3k
.ends inv
...
*subcircuit instance
x1 1 48 inv
.print v(x1.1)
.plot v(x1.1)
```

Specifies the instantiation of subcircuit inv with instance name x1 placed between nodes 1 and 48. Subcircuit node 1 is shown both printed and plotted.

```
*SUBCKT definition
.subckt inv 1 2
r1 1 3 rval
r2 3 4 rval1
r3 4 2 rval2
.ends inv
...
*subcircuit instance
x1 1 2 inv rval=6 rval2={rval1+1k}
.param rval1 = 10
x2 2 3 inv rval=4
.ic v(x1.1)=0
```

Specifies two instantiations of subcircuit inv with instance names x1 and x2. Parameters are assigned directly, using expressions and via the **`.PARAM`** command. Subcircuit node 2 is given the initial condition of 0 V via the **`.IC`** command.

Usually, pins of subcircuit (X) instances are supposed to be specified in the order corresponding to that of the **`.SUBCKT`** declaration (pins mapped by position). For instance, in the following:

```
.SUBCKT foo A B
..
.ends
X1 P1 P2 foo p1=1
```

then A is mapped to P1 and B to P2.

However, it is also possible to instantiate a subcircuit (X) using pin names of a **.SUBCKT** definition (pins mapped by name), providing that option **XBYNAME** is set. The example above can be rewritten as:

```
.option xbyname
.SUBCKT foo A B
..
.ends
X1 foo PARAM: p1=1 PIN: B=P2 A=P1
```

The following example shows how to instantiate a subcircuit (X) using pin names (pins mapped by name) and using net names (nets mapped by pin), providing that option **XBYNAME** is set:

```
.option XBYNAME
.subckt inv2 a y
X1 a net1 inv ln=0.18u wn=2.0u lp=0.18u wp=4.0u
X2 net1 y inv ln=0.18u wn=2.0u lp=0.18u wp=4.0u
.ends
X1 in out1 inv2
X2 MODEL: inv2 PIN: a=in y=out2
X3 MODEL: inv2 NET: in=a out3=y
XTOP in out inv2
```

The following example shows how the **DTEMP** parameter is used to specify a difference between the circuit temperature and the subcircuit instance temperature. In this example Eldo will assume that the temperature of devices inside X1 will be 40 + 10 = 50 °C:

```
.subckt r 1 2
r1 1 2 1 tc1=1
.ends
i1  1 0 1
x1 1 0 r dtemp = 10
.TEMP 40
.op
.end
```

Chapter 5
Sources

Introduction

Eldo provides a number of sources (stimuli generators) which can be divided into four groups as shown below:

Independent Sources

Two types of independent sources are provided:

Independent Voltage Source	V
Independent Current Source	I

Independent sources can be assigned a time-dependent value for transient analysis. The time zero values of time dependent sources are used for DC analysis. Eight types of time dependent source are provided:

Amplitude Modulation Function	AM
Exponential Function	EXP
Noise Function	NOISE
Noise Table Function	NOISE TABLE
Pattern Function	PATTERN
Pulse Function	PULSE
Piece Wise Linear Function	PWL
Single Frequency FM Function	SFFM
Sine Function	SIN
Trapezoidal Pulse With Bit Pattern Function	PBIT
Exponential Pulse With Bit Pattern Function	EBIT

Linear Dependent Sources

Four types of linear dependent sources are provided:

Linear Voltage Controlled Voltage Source ($v = e \cdot v$)	E
Linear Current Controlled Current Source ($i = f \cdot i$)	F
Linear Voltage Controlled Current Source ($i = g \cdot v$)	G
Linear Current Controlled Voltage Source ($v = h \cdot i$)	H

where `E`, `F`, `G` and `H` are constants representing voltage gain, current gain, transconductance and transresistance respectively.

Non-linear Dependent Sources

Four types of non-linear dependent sources are provided, defined by:

Non-linear Voltage Controlled Voltage Source ($v = f(v)$)	E
Non-linear Current Controlled Current Source ($i = f(i)$)	F
Non-linear Voltage Controlled Current Source ($i = f(v)$)	G
Non-linear Current Controlled Voltage Source ($v = f(i)$)	H

where the function is a polynomial and the arguments multi-dimensional. The polynomial functions are specified by the coefficients $p_0 \ldots p_n$. The significance of the coefficients depends upon the order of the polynomial, as shown below:

1st order polynomial

f_a is the function argument. The function value f_v is computed in the following manner:

$$f_v = P_0 + (P_1 \cdot f_a) + (P_2 \cdot f_a^2) + (P_3 \cdot f_a^3) + (P_4 \cdot f_a^4) + \ldots$$

2nd order polynomial

f_a and f_b are the function arguments. The function value f_v is computed in the following manner:

$$f_v = P_0 + (P_1 \cdot f_a) + (P_2 \cdot f_b) + (P_3 \cdot f_a^2) + (P_4 \cdot f_a \cdot f_b) + (P_5 \cdot f_b^2) + (P_6 \cdot f_a^3) + (P_7 \cdot f_a^2 \cdot f_b) + (P_8 \cdot f_a \cdot f_b^2) + \ldots$$

3rd order polynomial

f_a, f_b and f_c are the function arguments. The function value f_v is computed in the following manner:

$$f_v = P_0 + (P_1 \cdot f_a) + (P_2 \cdot f_b) + (P_3 \cdot f_c) + (P_4 \cdot f_a^2) + (P_5 \cdot f_a \cdot f_b) + (P_6 \cdot f_a \cdot f_c) + (P_7 \cdot f_b^2) + (P_8 \cdot f_b \cdot f_c) + (P_9 \cdot f_c^2) + (P_{10} \cdot f_a^3) + \ldots$$

The following pages contain syntax and parameter explanations, together with a number of worked examples, of each of the source options provided by Eldo.

S, Y, Z Parameter Extraction

A set of commands in Eldo allow the user to extract the S parameters (Scattering parameters), the Y parameters (Admittance) or the Z parameters (Impedance) in the frequency domain for a specified circuit. The circuit can have any number of ports. Special sources must be added at each port of the circuit to be analyzed.

See "S, Y, Z Parameter Extraction" on page 376.

Independent Sources

Independent Voltage Source

Independent Source Element

```
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]] [TIME_DEPENDENT_FUNCTION1]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val] [LPORT=val]
+ [MODE=keyword] [NOISETEMP=val]
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]] [TIME_DEPENDENT_FUNCTION1]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val]
```

Noise Source

```
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE [THN=VAL] [FLN=VAL]
+ [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL] [FMAX=VAL] [NBF=VAL]
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE [THN=VAL] [FLN=VAL]
+ [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL] [FMAX=VAL] [NBF=VAL]
```

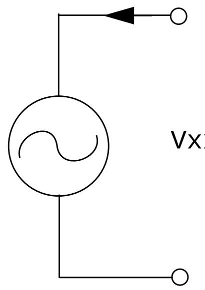
Tabular Noise Source

```
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG|HARM_DEC|HARM_OCT] [DB|MA]
+ (f1 val1) (f2 val2) ...
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG|HARM_DEC|HARM_OCT] [DB|MA]
+ (f1 val1) (f2 val2) ...
```

Multi-Tone Source

```
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val]
+ FOUR [DELAY=val] fund1 [fund2 [fund3]] MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
Vxx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string
+ [IPORT=val] [CPORT=val] [LPORT=val] [MODE=keyword] [NOISETEMP=val]
+ FOUR [DELAY=val] fund1 [fund2 [fund3]] MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
```

1. Refer to the **EXP**, **PATTERN**, **PULSE**, **PWL**, **SFFM** and **SIN** source functions.



Note

The current flows from the positive node `NP`, through the voltage source, to the negative node `NN`.

Parameters

- `xx`

  Independent voltage source name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `DCVAL`

  Value of the DC voltage.

- `ACMAG`

  AC magnitude in volts. Default value is 1.

- `ACPHASE`

  AC phase in degrees. Default value is zero.

- `TIME_DEPENDENT_FUNCTION`

  Refers to the time dependence of the voltage source (**PWL**, **PULSE**, **EXP**, **PATTERN**, **SSFM** and **SIN**). A multi-tone sine wave time dependence can also be defined with the **FOUR** keyword as detailed in the separate syntax. (See below for more details).

- **TC1**, **TC2**

  First and Second order temperature coefficients. Default values are zero. **TC2** can be specified even if **TC1** is not.

$$VVAL(T) = VAL(T_{nom})(1 + \mathrm{TC1}(T - T_{nom}) + \mathrm{TC2}(T - T_{nom}))$$

The above equation defines the value of the voltage (*VVAL*) as a function of temperature, where ***T*** is the operating temperature specified either by the **.TEMP** command, or the **T** parameter. ***Tnom*** is the nominal temperature for which the voltage source has voltage ***VAL***. Default value of ***Tnom*** is 27 °C and is adjustable using **.OPTION TNOM**.

- **NOISE**

  Generates a noise source.

> **Note**
> For the Noise Source, at least one parameter has to be specified after the **NOISE** keyword, otherwise Eldo issues the following message: "Incorrect number of parameters for noise source VNOISE"

- **THN**

  Defines the white noise level in A^2/Hz or V^2/Hz respectively. Can be specified as a bias-dependant expression.

- **FLN**

  Defines the 1/f or Flicker Noise level at 1 Hz in $A^2.Hz^{(1-alpha)}$ or $V^2.Hz^{(1-alpha)}$ respectively. Default is 0. Can be specified as a bias-dependant expression.

- **ALPHA**

  Frequency exponent for 1/f noise.

- **FC**

  Cut-off frequency of the low pass noise filter.

- **N**

  Filter order.

- **FMIN**

  Lower limit of the noise frequency band.

- **FMAX**

  Upper limit of the noise frequency band.

> **Note**
> **FMIN** and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion.

- **NBF**

  Specifies the number of sinusoidal sources with randomly distributed amplitude and phase from which the noise source is composed. Default is 50.

Note

Parameters **FMIN**, **FMAX** and **NBF** of Noise Sources are taken into account for Transient analysis only.

Please refer to ".NOISETRAN" on page 747 for further information.

- **NOISE TABLE**

  Keyword indicating that the noise source has a tabular description. See "Noise Table Function" on page 330.

- **INTERP**

  Specifies how to interpolate between different frequency values: **LIN** means linear interpolation; **LOG**, **OCT** or **DEC** are all used in the same sense which is logarithmic, octal, or decimal interpolation; **HARM_DEC** or **HARM_OCT** specify a logarithmic or octal interpolation around each harmonic of the **.SSTNOISE** analysis.

- `f1`, `f2`

  Frequency values in Hertz.

- `val1`, `val2`

  Values in V^2/Hz.

- **FOUR**

  Keyword specifying that the source is multi-tone.

- **DELAY**=`val`

  Specifies the time (seconds) that the output is delayed by. This parameter is only effective in **.TRAN**. Default value is zero.

- `fund1 [fund2 [fund3]]`

  Parameters to define fundamental frequencies. A source can be defined with up to 3 tones.

- **MA|RI|DB|PMA|PDB|PDBM**

  Keyword defining the format (**MA**—Magnitude Angle, **RI**—Real Imaginary, **DB**—Magnitude in dB Angle, **PMA**—Power in Watt Angle, **PDB**—Power in dB Angle, **PDBM**—Power in dBm Angle). Power formats (**PMA**, **PDB** and **PDBM**) are only allowed on port sources (voltage or current sources where **IPORT** is specified). For multi-tone, the format is used in conjunction with the `real_val1`, `real_val2` specification below. For tabular (only **DB|MA**), values are couples `f1 val1`, see above.

- `int_val1`

  Defines the index of the harmonic according to `fund1`.

- `int_val2`

  Defines the index of the harmonic according to `fund2`.

- `int_val3`

  Defines the index of the harmonic according to `fund3`.

  The group of 1 to 3 index values define a frequency. For example, for a source with 3 fundamental frequencies, (`int_val1`, `int_val2`, `int_val3`) specifies the frequency:

  ```
  F = int_val1*fund1+int_val2*fund2+int_val3*fund3
  ```

- `real_val1`, `real_val2`

  Defines the complex value of the source for the corresponding frequency in the specified format (MA, RI or DB).

- **`RPORT`**

  Resistance of the port value which defaults to 50 Ω whenever **`RPORT_TC1`**, **`RPORT_TC2`** or **`IPORT`** is specified. The possibility of having **`NONOISE`** on this resistance is allowed.

- **`ZPORT_FILE`**

  Specifies the Touchstone file name that contains the complex port impedance.

- **`NONOISE`**

  Used in conjunction with **`RPORT`**. Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the port resistance presents no noise contribution to the noise analysis.

- **`NOISE`**`=1`

  Specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **`NONOISE`** specification on a **`.MODEL`** card.

- **`RPORT_TC1`** & **`RPORT_TC2`**

  Temperature coefficients of **`RPORT`**: they both default to 0. If **`RPORT`** happens to be 0 (in parametric simulations for instance), there will be no S extraction performed.

- **`IPORT`**

  This is a strictly positive number that is unique and is used as the port number: this number is used for naming the outputs (for instance, `.PLOT AC S(1,2)`). An error message will be issued if two port instances have the same value for **`IPORT`**, or if an **`IPORT`** is missing (for example maximum **`IPORT`** number found in the netlist is 4, and there is no instance with **`IPORT`** 3).

- **`CPORT`**

  Capacitor placed in series with **`RPORT`**. Defaults to 0, in which case it behaves like a zero voltage source (i.e. **`CPORT`** has no effect).

- **LPORT**

  Inductor placed in series with **RPORT**. Defaults to 0.

- **MODE**=**SINGLE|COMMON|DIFFERENTIAL**

  Mixed-mode S parameter selection.
  SINGLE specifies the port as single ended, it is dedicated to S parameter extraction. Default.
  COMMON and **DIFFERENTIAL** specify that the port is not single ended. Such ports are split into two linked sources that are either common (same amplitude and same phase) or differential (same amplitude but opposite phases). If the S parameters are extracted a "non-single ended" port is equally common and differential depending on which display is required. During simulation (DC, AC or TRAN) this port is either common or differential depending on the specified mode keyword.

- **NOISETEMP**

  Corresponds to the temperature value used for the calculation of the NOISE generated by **RPORT**. The default is the temperature of the circuit, but should be set to 16.85 to comply with IEEE specifications. When this parameter is specified, it overrides the **INPUT_TEMP** parameter in the **.SNF** command.

Examples

```
vplus n12 n13 24
```

Specifies a fixed voltage of 24V between nodes n12 and n13.

```
v7 n4 n9 dc 1.2 ac 1.0e-3
```

Specifies a DC voltage of 1.2V placed between nodes n4 and n9 and an AC voltage of 1mV.

```
v7 n4 n9 ac 1.2 pwl (0 3 5n 0 10n 0)
```

Specifies an AC voltage source of 1.2V together with a Piece Wise Linear (**PWL**) voltage source definition placed between nodes n4 and n9.

```
V1  n1  n2  FOUR  fund1 MA
+   (0)  5  0
+   (1)  2.5  -90
.param fund1=100meg
```

The above example defines a source having the following time dependence:

```
V1(t) = 5 + 2.5*sin(2*pi*100meg*t)
      V2  n3  n4  FOUR fund1   fund2  DB
      +          (0, 0)   0    0
      +          (1, 0)   6   -90
      +          (0, 1)  -6    0
      +          (-2, 2)  0    45
      .param fund1=900meg  fund2=1.2giga
```

The value of V2 will be computed as follows:

```
V2 = A + B*sin(2*pi*900e6*t) + C*cos(2*pi*1.2e9)
         + A*cos((-2*900e6 + 2*1.2e9)*2*pi*t + PI/4)
```

with:

`A` is computed as 0dB from 1V, i.e. `A` is 1V
`B` is 6dB with reference to 1V, then it is roughly 2.0V
`C` is -3dB with reference to 1V, then it is roughly 0.5V
The `PI`/4 term at the end of the expression corresponds to 45 degrees, as specified in the SPICE card.

Note

Independent voltage sources defined to have a voltage of 0V may be removed by Eldo in order to simplify calculations. If you wish to use a voltage source as a current probe, use **`.OPTION AMMETER`** to prevent Eldo from removing such voltage sources defined to have a voltage of 0V. Moreover, currents through components may be measured directly, therefore, the use of voltage sources as current probes is not usually necessary.

The next example specifies a current source of 1A between nodes 1 and 0. It has a first order temperature coefficient (TC1) of 1. The second order temperature coefficient (TC2) will default to 0.

```
v1 1 0 1 tc1=1
```

The next example shows how **`THN`** and **`FLN`** parameters can be specified as bias-dependant expressions:

```
vx p1 p2 NOISE
+ FLN = 'kf_mod*(pow(abs(i(vx)),af))'
+ THN = '(4*K*(temper + 273))/(abs(v(p1,p2)/i(vx)))'
```

Independent Current Source

Independent Source Element

```
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]] [TIME_DEPENDENT_FUNCTION1]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val] [LPORT=val]
+ [NOISETEMP=val]
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]] [TIME_DEPENDENT_FUNCTION1]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val]
```

Noise Sources

```
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]] [RPORT_TC1=val]
+ [RPORT_TC2=val] [IPORT=val] [CPORT=val] [LPORT=val] [NOISETEMP=val]
+ NOISE [THN=VAL] [FLN=VAL] [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL]
+ [FMAX=VAL] [NBF=VAL]
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [IPORT=val] [CPORT=val]
+ ZPORT_FILE=string [LPORT=val] [MODE=keyword] [NOISETEMP=val]
+ NOISE [THN=VAL] [FLN=VAL] [ALPHA=VAL] [FC=VAL] [N=VAL] [FMIN=VAL]
+ [FMAX=VAL] [NBF=VAL]
```

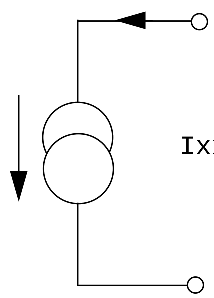
Tabular Noise Source

```
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG|HARM_DEC|HARM_OCT] (f1 val1) (f2 val2) ...
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val] NOISE TABLE
+ [[INTERP=]DEC|OCT|LIN|LOG|HARM_DEC|HARM_OCT] (f1 val1) (f2 val2) ...
```

Multi-Tone Source

```
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] [RPORT=val [NONOISE] [NOISE=1]]
+ [RPORT_TC1=val] [RPORT_TC2=val] [IPORT=val] [CPORT=val]
+ [LPORT=val] [NOISETEMP=val]
+ FOUR [DELAY=val] fund1 [fund2 [fund3]] MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
Ixx NP NN [[DC] DCVAL] [AC [ACMAG [ACPHASE]]]
+ [TC1=val] [TC2=val] ZPORT_FILE=string [IPORT=val] [CPORT=val]
+ [LPORT=val] [MODE=keyword] [NOISETEMP=val]
+ FOUR [DELAY=val] fund1 [fund2 [fund3]] MA|RI|DB|PMA|PDB|PDBM
+ (int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2
+ {(int_val1 [,int_val2 [,int_val3]]) real_val1 real_val2}
```

1. Refer to the **EXP**, **PULSE**, **PWL**, **SFFM** and **SIN** source functions.



Note

The current flows from the positive node NP, through the current source, to the negative node NN.

Parameters

- xx

  Independent current source name.

- NP

  Name of the positive node.

- NN

  Name of the negative node.

- DCVAL

  Value of the DC current source in amperes.

- ACMAG

  AC magnitude in amperes. Default value is 1.

- ACPHASE

  AC phase in degrees. Default value is 0.

- TIME_DEPENDENT_FUNCTION

  Refers to the time dependence of the voltage source (**PWL**, **PULSE**, **EXP**, **PATTERN**, **SSFM** and **SIN**). A multi-tone sine wave time dependence can also be defined with the **FOUR** keyword as detailed in the separate syntax. (See below for more details).

- **TC1**, **TC2**

  First and Second order temperature coefficients. Default values are zero. **TC2** can be specified even if **TC1** is not.

$$IVAL(T) = VAL(T_{nom})(1 + \mathrm{TC1}(T - T_{nom}) + \mathrm{TC2}(T - T_{nom})^2)$$

The above equation defines the value of the current (*IVAL*) as a function of temperature, where ***T*** is the operating temperature specified either by the **.TEMP** command, or the **T** parameter. ***Tnom*** is the nominal temperature for which the current source has current ***VAL***. Default value of ***Tnom*** is 27 °C and is adjustable using **.OPTION TNOM**.

- **NOISE**

  Generates a noise source.

Note

For the Noise Source, at least one parameter has to be specified after the **NOISE** keyword, otherwise Eldo issues the following message: “Incorrect number of parameters for noise source INOISE”

- **THN**

  Defines the white noise level in A^2/Hz or V^2/Hz respectively. Can be specified as a bias-dependant expression.

- **FLN**

  Defines the 1/f or Flicker Noise level at 1 Hz in $A^2.Hz^{(1-alpha)}$ or $V^2.Hz^{(1-alpha)}$ respectively. Default is 0. Can be specified as a bias-dependant expression.

- **ALPHA**

  Frequency exponent for 1/f noise.

- **FC**

  Cut-off frequency of the low pass noise filter.

- **N**

  Filter order.

- **FMIN**

  Lower limit of the noise frequency band.

- **FMAX**

  Upper limit of the noise frequency band.

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion.

- **NBF**

  Specifies the number of sinusoidal sources with randomly distributed amplitude and phase from which the noise source is composed. Default is 50.

Note

Parameters **FMIN**, **FMAX** and **NBF** of Noise Sources are taken into account for Transient analysis only.

Please refer to ".NOISETRAN" on page 747 for further information.

- **NOISE TABLE**

  Keyword indicating that the noise source has a tabular description. See "Noise Table Function" on page 330.

- **FOUR**

  Keyword specifying that the source is multi-tone.

- **DELAY**=val

  Specifies the time (seconds) that the output is delayed by. This parameter is only effective in **.TRAN**. Default value is zero.

- fund1 [fund2 [fund3]]

  Parameters to define fundamental frequencies. A source can be defined with up to 3 tones.

- **MA|RI|DB|PMA|PDB|PDBM**

  Keyword defining the format (**MA**—Magnitude Angle, **RI**—Real Imaginary, **DB**—Magnitude in dB Angle, **PMA**—Power in Watt Angle, **PDB**—Power in dB Angle, **PDBM**—Power in dBm Angle). Power formats (**PMA**, **PDB** and **PDBM**) are only allowed on port sources (voltage or current sources where **IPORT** is specified). The format is used in conjunction with the real_val1, real_val2 specification below.

- int_val1

  Defines the index of the harmonic according to fund1.

- int_val2

  Defines the index of the harmonic according to fund2.

- int_val3

  Defines the index of the harmonic according to fund3.

  The group of 1 to 3 index values define a frequency. For example, for a source with 3 fundamental frequencies, (int_val1, int_val2, int_val3) specifies the frequency:

  ```
  F = int_val1*fund1+int_val2*fund2+int_val3*fund3
  ```

- real_val1, real_val2

  Defines the complex value of the source for the corresponding frequency in the specified format (MA, RI or DB).

- **RPORT**

  Resistance of the port value which defaults to 50 Ω whenever **RPORT_TC1**, **RPORT_TC2** or **IPORT** is specified. The possibility of having **NONOISE** on this resistance is allowed.

- **ZPORT_FILE**

  Specifies the Touchstone file name that contains the complex port impedance.

- **NONOISE**

  Used in conjunction with **RPORT**. Specifies that no noise model will be used for this device when performing noise analysis. Therefore, the port resistance presents no noise contribution to the noise analysis.

- **NOISE**=1

  Specifies that a noise model will be used for this device when performing noise analysis. Therefore, the device presents noise contribution to the noise analysis. This has precedence over any **NONOISE** specification on a **.MODEL** card.

- **RPORT_TC1** & **RPORT_TC2**

  Temperature coefficients of **RPORT**: they both default to 0. If **RPORT** happens to be 0 (in parametric simulations for instance), there will be no S extraction performed.

- **IPORT**

  This is a strictly positive number that is unique and is used as the port number: this number is used for naming the outputs (for instance, `.PLOT AC S(1,2)`). An error message will be issued if two port instances have the same value for **IPORT**, or if an **IPORT** is missing (for example maximum **IPORT** number found in the netlist is 4, and there is no instance with **IPORT** 3).

- **CPORT**

  Capacitor placed in parallel to **RPORT**. Defaults to 0, in which case it behaves like a zero voltage source (i.e. **CPORT** would have no effect).

- **LPORT**

  Inductor placed in series with **RPORT**. Defaults to 0.

- **MODE**=**SINGLE|COMMON|DIFFERENTIAL**

  Mixed-mode S parameter selection.
  SINGLE specifies the port as single ended, it is dedicated to S parameter extraction. Default.
  COMMON and **DIFFERENTIAL** specify that the port is not single ended. Such ports are split into two linked sources that are either common (same amplitude and same phase) or differential (same amplitude but opposite phases). If the S parameters are extracted a “non-single ended” port is equally common and differential depending on which display is required. During simulation (DC, AC or TRAN) this port is either common or differential depending on the specified mode keyword.

- **`NOISETEMP`**

  Corresponds to the temperature value which will be used for the calculation of the NOISE generated by **`PORT`**. The default is the temperature of the circuit, but should be set to 16.85 to comply with IEEE specifications. When this parameter is specified, it overrides the **`INPUT_TEMP`** parameter in the **`.SNF`** command.

Examples

```
i23 n2 n3 1.0e-4
```

Specifies a 0.1mA current flowing from node n2 towards node n3.

```
i41 n2 n4 dc 1.0e-3 ac 1.0e-6 45
```

Specifies a DC current of 1mA flowing from node n2 towards node n4 and an AC current of 1.0×10^{-6}A with a phase of 45 degrees.

```
i1 1 0 1 tc1 = 1
```

Specifies a current source of 1A between nodes 1 and 0. It has a first order temperature coefficient of 1. The second order temperature coefficient (TC2) will default to 0.

The next example shows how **`THN`** and **`FLN`** parameters can be specified as bias-dependant expressions:

```
ix p1 p2 NOISE
+ FLN = 'kf_mod*(pow(abs(i(vx)),af))'
+ THN = '(4*K*(temper + 273))/(abs(v(p1,p2)/i(vx)))'
```

Amplitude Modulation Function

AM (AMPLITUDE OFFSET FM FC TD**)**

Generates a time-dependent Amplitude Modulated signal. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Parameters

- AMPLITUDE

  Signal amplitude. Default is 0.

- OFFSET

  Offset of the signal. Default is 0.

- FM

  Modulation frequency. Default is 1/TSTOP.

- FC

  Carrier frequency. Default is 0.

- TD

  Delay before signal starts. Default is 0.

The waveform is described by:

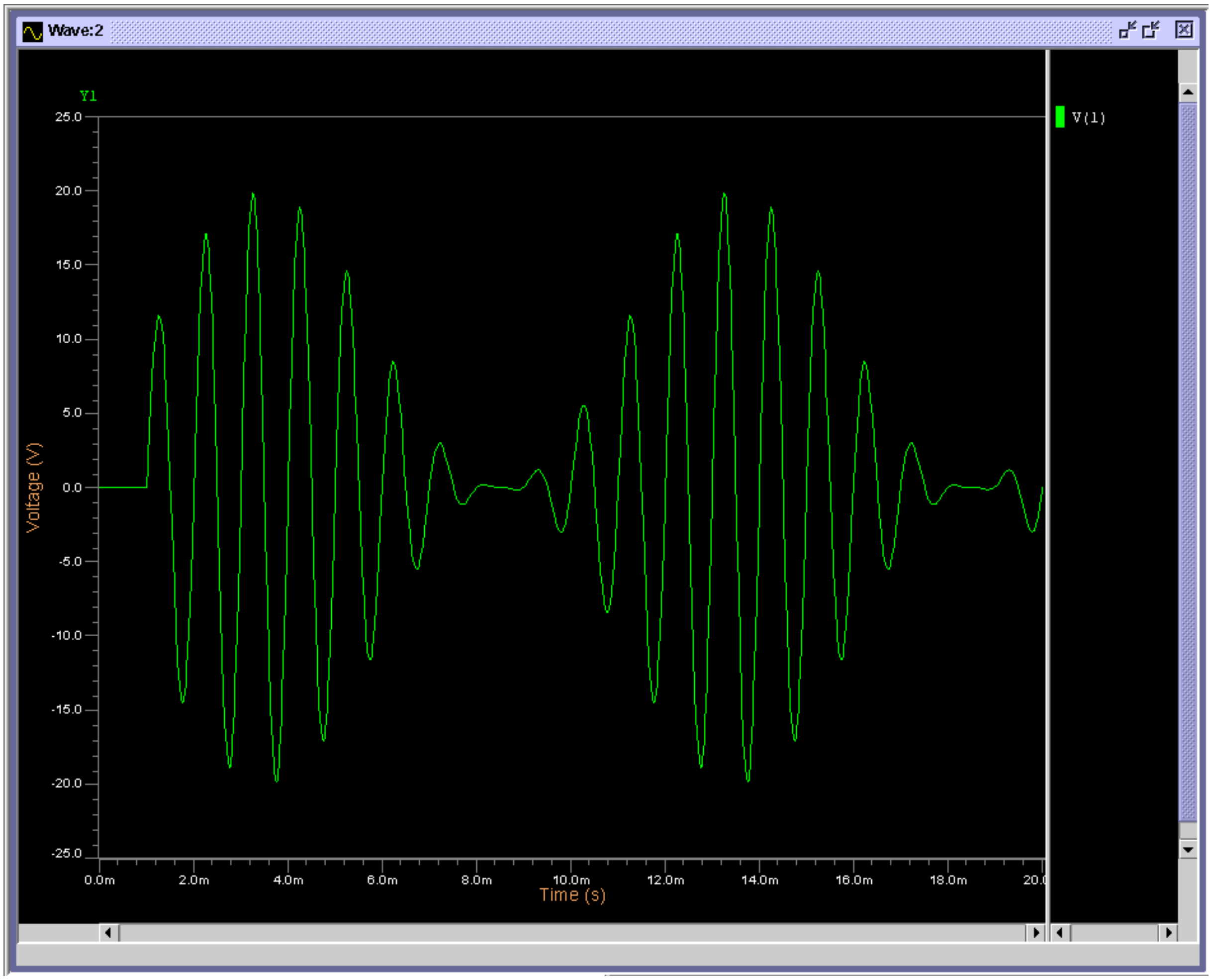
$$V = AMPLITUDE \times (OFFSET + \sin(2\pi \times FM \times (time - TD))) \times \sin(2\pi \times FC \times (time - TD))$$

Example

```
vam 1 0 am (10 1 p100 1k 1m)
r1 1 0 1
.tran 1m 20m
.plot tran v(1)
.param p100 = 100
.end
```

The diagram below, has been generated by the above netlist. It shows amplitude modulation for a signal with amplitude 10 and an offset of 1. There is a signal delay of 1 millisecond and the modulation frequency and carrier frequency is set at 100 and 1000 respectively.

Figure 5-1. AM Function Example



Exponential Function

```
EXP (V1 V2 [TD1 [TAU1 [TD2 [TAU2]]]])
```

Generates an exponentially damped pulse as defined below. To be used in combination with independent voltage (`V`xx) or current (`I`xx) sources.

Parameters

- `V1`

  Initial value in volts or amperes.

- `V2`

  Asymptotic, or target value of the pulse in volts or amperes.

- `TD1`

  Rise delay time in seconds. Default value is zero.

- `TAU1`

  Rise time constant in seconds. Default value is `TPRINT`.

- `TD2`

  Fall delay time in seconds. Default value is `TD1+TPRINT`.

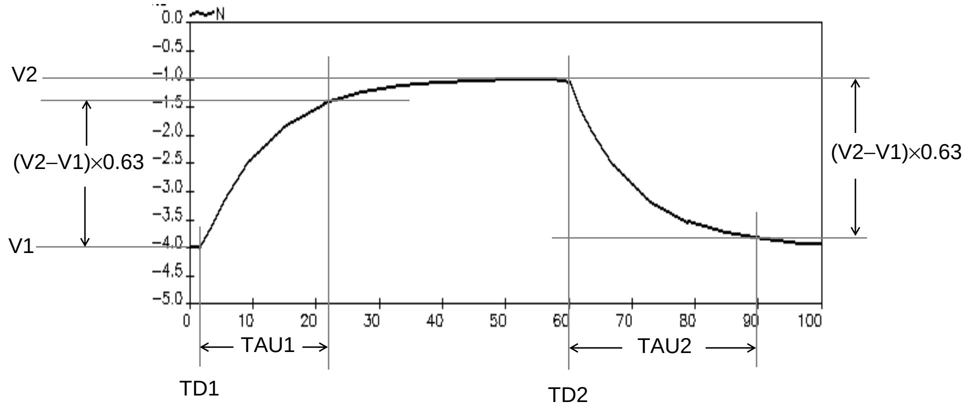
- `TAU2`

  Fall time constant in seconds. Default value is `TPRINT`.

The waveform is described by the following relationships:

Time	Voltage
0 to TD1	V1
TD1 to TD2	$V1 + (V2 - V1)\left(1 - \exp\frac{-(time - TD1)}{TAU1}\right)$
TD2 to TSTOP	$V1 + (V2 - V1)\left(1 - \exp\frac{-(time - TD1)}{TAU1}\right) + (V1 - V2)\left(1 - \exp\frac{-(time - TD2)}{TAU2}\right)$

Figure 5-2. Exponential Function



Example

```
vin n3 0 exp (-4 -1 2n 10n 60n 10n)
```

Specifies a voltage source vin between node n3 and ground. The time dependent voltage is described by:

Time	Voltage
0ns to 2ns	–4V.
2ns to 60ns	Exponential rise from –4V to –1V with the rise time constant 10ns.
60ns – **TSTOP**	Exponential fall from –1V to –4V with the fall time constant 10ns. The **TSTOP** value is set in the **.TRAN** command.

Noise Function

```
NOISE THN FLN ALPHA [FC N] [FMIN] [FMAX] [NBF] [value={expr}]
```

Generates a noise source and is used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

> **Note**
> This source is effective only during **.NOISE** and **.NOISETRAN** analyses. A noise source has no effect during an AC or Transient simulation.

You can define correlation coefficients between two independent noise sources with the command “.NOISE_CORREL” on page 746.

Parameters

- THN

  Defines the white noise level in A^2/Hz or V^2/Hz respectively. Default is 0.

- FLN

  Defines the 1/f or Flicker Noise level at 1 Hz in $A^2.Hz^{(1\text{-alpha})}$ or $V^2.Hz^{(1\text{-alpha})}$ respectively. Default is 0.

- ALPHA

  Frequency exponent for 1/f noise. Default is 1.

- FC

  Cut-off frequency of the low pass noise filter.

- N

  Filter order.

- **FMIN**

  Lower limit of the noise frequency band.

- **FMAX**

  Upper limit of the noise frequency band.

- **NBF**

  Specifies the number of sinusoidal sources with randomly distributed amplitude and phase from which the noise source is composed. Default is 50.

- **value**={expr}

  Defines an expression function of (noise) frequency.

Note

`FMIN` and **`FMAX`** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**`FMIN`**, **`FMAX`**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion.

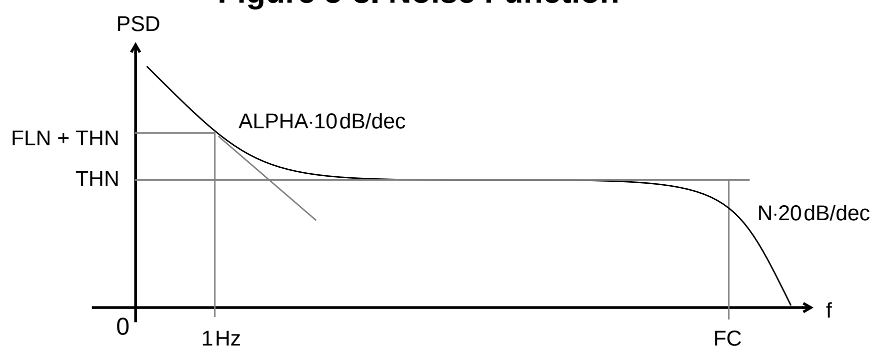
The Power Spectral Density (PSD) may be described as:

$$S_x = \left(THN + \frac{FLN}{f^{\ ALPHA}}\right)\left(\frac{1}{(1+\Omega^2)^N}\right)$$

where:

$$\Omega = \frac{f}{FC}$$

Figure 5-3. Noise Function



Examples

```
v1 n1 n2 noise 1e-17 0 1
```

Specifies a voltage noise source v1 placed between nodes n1 and n2. The White Noise level of the PSD has a value of $1\times10^{-17} V^2$/Hz, the Flicker Noise level is 0 and the frequency exponent for 1/f noise is 1.

```
i5 n5 0 noise 1e-20 1e-15 1.3 1meg 2
```

Specifies a current noise source i5 placed between node n5 and ground. The White Noise level of the PSD has a value of $1\times10^{-20} A^2$/Hz, the Flicker Noise level is 1×10^{-15},the frequency exponent for 1/f noise is 1.3, the Cut-off frequency of the low pass filter is 1MHz and the order of the filter is 2.

```
.param _pi=3.1415 p_fref=13e6

VN1 VDD 0 5 noise
+ value = { 2*_PI*sin(FREQ/p_fref) }
```

```
.subckt DSQN outp outn ref param: fref=26meg ds_order=3 nscale=1.0
.param p_kds = '2*pi*2*pi/(12*fref)'
.param tom1='2*(ds_order-1)'
Vn outp outn noise
+ value ={nscale * pkds * pwr(2*abs(sin(pi * FREQ / fref)), tom1)}
.ends
```

Defines an expression function of (noise) frequency.

Noise Table Function

```
NOISE TABLE [[INTERP=]DEC|OCT|LIN|LOG|HARM_DEC|HARM_OCT] [DB|MA]
+ (f1 val1) (f2 val2) ...
```

Generates a noise source with tabular description and is used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Note

This source is effective only during **.NOISE** and **.SSTNOISE** analyses. A noise source has no effect during other analysis type simulations, a zero voltage/current source will be assumed.

Specify input noise source values depending on the frequency via the **TABLE** keyword. Eldo will assume zero current source or zero voltage source in any mode other than **NOISE** (for DC, AC, TRAN, and so on).

Parameters

- **NOISE TABLE**

  Keyword indicating that the noise source has a tabular description.

- **INTERP**

  Specifies how to interpolate between different frequency values: **LIN** means linear interpolation; **LOG**, **OCT** or **DEC** are all used in the same sense: logarithmic, octal, or decimal interpolation; **HARM_DEC** or **HARM_OCT** specify a logarithmic or octal interpolation around each harmonic of the **.SSTNOISE** analysis.

- f1, f2

  Frequency values in Hz.

- val1, val2

  Corresponds to the noise Power Spectral Density of the source at frequency f1, f2 (in V^2/Hz or A^2/Hz). Noise is specified relative to the signal.

- **MA|DB**

  Keyword defining the format: **MA**—Magnitude, **DB**—Magnitude in dB.

Examples

```
v1 n1 n2 noise table interp=log
+ 1.0000E+00 'p1*pwr(2*abs(sin(pi* 1.0000E+00 / pref)), 6)'
+ 1.0965E+00 'p1*pwr(2*abs(sin(pi* 1.0965E+00 / pref)), 6)'
+ 1.2023E+00 'p1*pwr(2*abs(sin(pi* 1.2023E+00 / pref)), 6)'
+ 1.3183E+00 'p1*pwr(2*abs(sin(pi* 1.3183E+00 / pref)), 6)'
+ 1.4454E+00 'p1*pwr(2*abs(sin(pi* 1.4454E+00 / pref)), 6)'
+ 1.5849E+00 'p1*pwr(2*abs(sin(pi* 1.5849E+00 / pref)), 6)'
+ 1.7378E+00 'p1*pwr(2*abs(sin(pi* 1.7378E+00 / pref)), 6)'
...
```

Specifies a tabular voltage noise source v1 between nodes n1 and n2, with a logarithmic interpolation around each frequency of the **.NOISE** analysis.

Pattern Function

```
PATTERN VHI VLO TDELAY TRISE TFALL TSAMPLE BITS RB=val R[=val]
PATTERN_FILE=filename
```

Generates a pulsating (digital like) source whose sequential values are defined as a distinct series of 1 and 0 values. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources. Periodic patterns can also be specified. Parameters of a PATTERN source can be modified with a **.STEP** command.

A pattern can be written inside an external file and included in the function. It can be described in csv format, SPICE-like format, or blank separator format.

An alternative compact pattern syntax is also available that allows easier user-input of patterns.

Parameters

- VHI

  Voltage representing the 1 string value for **PATTERN** sources.

- VLO

  Voltage representing the 0 string value for **PATTERN** sources.

- TDELAY

  Delay before the **PATTERN** series is started. The value assigned during this time is the first string value of the series.

- TRISE

  Rise time between **PATTERN** values in seconds.

- TFALL

  Fall time between **PATTERN** values in seconds.

- TSAMPLE

  Time spent at 1 or 0 **PATTERN** value.

- BITS

  String of 1, 0 and Z values representing a **PATTERN** source. This can also be specified via a parameter using $(PAT) where PAT is the parameter as specified in the **.PARAM** statement. Z means high impedance, and this releases the applied signal, i.e. when the Z state is active, the signal will be disconnected from the node, and the node will be computed by Eldo as if it were a non-input signal.

  An alternative compact pattern syntax is also available that allows easier user-input of patterns. The syntax is defined by the following rules:

  BitPattern =>
  [*BitPattern* ^ *number_of_bits***]** | **[***BitPattern***]** | *BaseElement BitPattern* | *BaseElement*

  BaseElement => **0** | **1** | **Z**

where *number_of_bits* is a numerical value lower than 100. You can change this limit using the option "PATTERN_MAX_ALLOWED_COEFF" on page 959.

- **RB**=val

  Specifies from which bit the pattern is repeated. Used when R is -1 or positive.

- **R**[=val]

  Specifies the pattern is periodic.

  If R=-1, the pattern is repeated continuously (identical to R without any value).

  If R=0, the pattern will not be repeated (identical to no R keyword).

  If R= a positive value x, the pattern is repeated x times.

- **PATTERN_FILE**=filename

  Include a pattern written inside an external file. The pattern file must contain all the information about the pattern and can use parameters defined in the netlist. It can be described in any of the following ways:

  - csv format, for example:

    ```
    Amplitude, 0, 100p , 100p, 100p, 10n, 1100101011100011 00101
    ```

  - SPICE-like format, for example:

    ```
    Amplitude 0
    *comment
    + 100p 100p 100p 10n !comment
    #com
    comment...
    ...
    #endcom
    + $(PPattern)
    ```

  - blank separator format, for example:

    ```
    Amplitude 0 100p 100p 100p 10n $(PPattern)
    ```

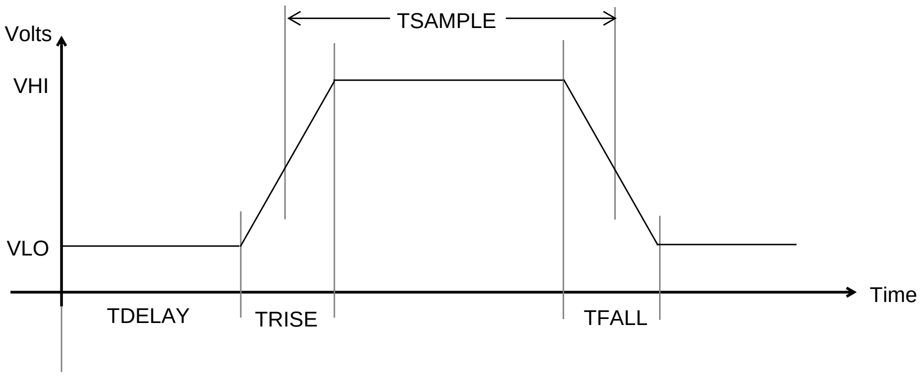
  - split among several lines with the bits pattern also split, for example:

    ```
    Amplitude, 0,
    100p, 100p, 100p,
    10n,
    1100
    1010
    1110
    0011
    00101
    ```

  Known limitations for pattern file specification:

  - There can only be one pattern specified in a file
  - Any errors in the pattern file will be reported as if it was in the netlist file

Figure 5-4. Pattern Function



Examples

```
v3 20 0 pattern 5 0 1u 2u 2u 8u 110010101 R
```

Specifying `R` at the end of the pattern means that it is periodic. Therefore, the equivalent pattern here is: `11001010111001010111001010111001010111 0.....`

```
v1 1 2 pattern 5 0 10n 5n 10n 20n 001111000
```

Specifies a DC voltage source placed between nodes 1 and 2 defined by a string of 1 and 0 values. The voltage representing the 0 string value is 0 V and the voltage level representing the 1 voltage level is 5 V. Rise and fall times of the voltage source are 5 and 10 ns respectively. The voltage source has the following characteristics:

1. Between time 0 and 10 ns, the voltage source remains at 0 V (`delay=10n`).
2. The voltage source remains at 0 V for 2×20 ns (`tbit=20n`).
3. The voltage source rises to a 5 V level in 5 ns (`trise=5n`).
4. The voltage source remains at 5 V for 4×20 ns.
5. The voltage level falls to 0 V in 10 ns (`tfall=10n`).
6. The voltage level remains at 0 V for 3×20 ns.

```
VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN {amp} {-amp}
+ 100p 100p 100p 10n
+ 0111111001111101111111011111100000
+ 00011111110101010111111110011001111
.PARAM amp=400m
.tran 0 1u
.plot tran v(input_pattern)
.step P(amp) 400m 0 50m
.end
```

The above example shows how to sweep parameters of a PATTERN source function using the **.STEP** command.

```
VINP input_pattern 0 PATTERN_FILE=pat.csv DC 0.0 AC 1 0.0
```

The above example shows how to specify a pattern file. The file *pat.csv* contains a pattern described in csv format.

The following show some examples of valid patterns using the alternative compact syntax, the first two specifications are equivalent:

```
VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN 100m 0 0 100p 100p 10n
[[100]^4][1^3][0^3] R

VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN 100m 0 0 100p 100p 10n
100100100100111000 R
```

The following two specifications are also equivalent:

```
VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN 100m 0 0 2n 140p 5n 00
[[[1^2][0^2]]^4] 1010 [1^3][0^3] 1100101

VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN 100m 0 0 2n 140p 5n 00
110011001100110010101110001100101
```

The following shows how to define a bit pattern with the alternative compact syntax, with the number of bits exceeding the defult 100:

```
.option PATTERN_MAX_ALLOWED_COEFF = 102
VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN 100m 0 0 10p 10p 500p
[[[1^101][0^2]^4]] 1010 [1^3] 1100101 R
```

The following shows the alternative compact syntax specified as a parameter:

```
.PARAM PPattern ="0[[[1^2][0^2]]^2]1010[1^3][0^3]1100101"
VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN 100m 0 100p 2n 140p 5n
$(PPattern) R
```

The following shows another example of the alternative compact syntax:

```
VINP input_pattern 0  DC 0.0 AC 1 0.0 PATTERN Amplitude 0 0 10p 10p
500p  [[  [1 ^ 101]  [0 ^  2]  ^  4  ]  ] 1010 [1^3] 1100101 R
```

The following example indicates that the pattern should be repeated for ever from bit number 3 (that is, at 3n):

```
v1   1 0 pat (1 0 0.0n 0.1n 0.1n 1n b101101 rb=3 r=-1)
```

The following example indicates that the pattern should be repeated twice from bit number 3 (that is, at 3n)

```
v1   1 0 pat (1 0 0.0n 0.1n 0.1n 1n b101101 rb=3 r=2)
```

Pulse Function

```
PULSE (V0 V1 [TD [TR [TF [PW [PER]]]]])
```

Generates a periodic pulse as described below. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Parameters

- V0

  Initial value of DC voltage or current.

- V1

  Pulse magnitude in volts or amperes.

- TD

  Delay time in seconds. Default value is zero.

- TR

  Rise time in seconds. Default value is TPRINT.

- TF

  Fall time in seconds. Default value is TPRINT.

- PW

  Pulse width in seconds. Default value is TSTOP.

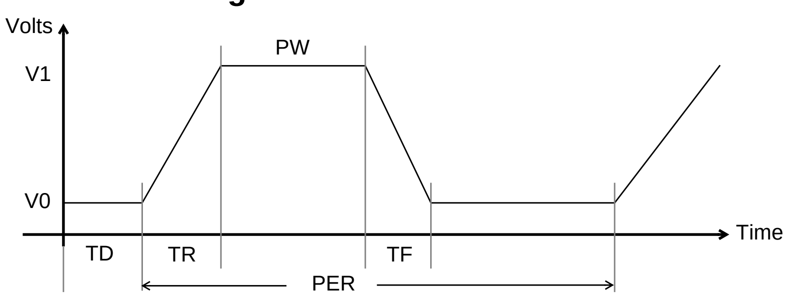
- PER

  Pulse period in seconds. Default value is TSTOP.

Note

If the parameters TR and TF are both set to 0, they are assigned a value of **TPRINT**, the time interval used for the printing of results of the transient analysis, defined in the **.TRAN** command.

Figure 5-5. Pulse Function



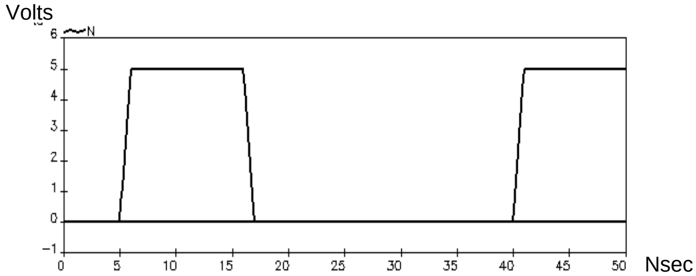
Example

```
vp n12 n13 pulse(0 5 5n 1n 1n 10n 35n)
```

Specifies the voltage source vp placed between the node n12 and n13. The voltage source has the following characteristics:

1. 0V from 0 to 5ns (td=5n, v0=0).
2. Rise from 0 to 5V in the time period 5 to 6ns (tr=1n, v1=5).
3. Remain at 5V from 6 to 16ns (pw=10n).
4. Fall from 5 to 0V in the time period 16 to 17ns (tf=1n).
5. 0V from 17 to 35ns (per=35n).
6. Second cycle starting at 35ns.

Figure 5-6. Voltage Source Characteristics



Piece Wise Linear Function

```
PWL (T1 V1 {TN VN} [TD=val] [R=val|PWLPERIOD=val] [SHIFT=val] [R]
+ [SCALE=val] [STRETCH=val])
PWL (FILE=pwl_file [TD=val] [R=val|PWLPERIOD=val] [SHIFT=val] [R]
+ [SCALE=val] [STRETCH=val])
PWL (FILE=pwl_file [COL=val] [ISTEP=val] [ISTART=val] [ISTOP=val]
+ [TD=val] [R=val|PWLPERIOD=val] [SHIFT=val] [SCALE=val] [STRETCH=val]
+ [R])
```

Generates a Piece Wise Linear function using straight lines between specified voltage points until `TN` is reached. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

The second two syntaxes above show how Eldo can read PWL corner points from a file, with the last syntax supporting multi-column files.

SHIFT=val and **R**=val can be used together in the same PWL statement. The keyword **R** alone (without a value specified) can be used in conjunction with **SHIFT**=val, but must be placed at the end of the statement.

The `TN VN` pairs can be separated by spaces or commas.

Note

`TD` and `SHIFT` are synonymous.

Parameters

- `V1,... VN`

  Value of the source at time ***Ti*** in volts or amperes. The source value at intermediate times is provided by linear interpolation. A high-impedance can be specified with the **Z** keyword.

- `T1,... TN`

  Time in seconds, at which ***Vi*** is supplied, where $Ti < Ti+1$.

- **TD**=VAL

  Delay time in seconds. Default value is zero.

Note

If the rise or fall times are 0, they are assigned a value of **TPRINT**, the time interval used for the printing of results of the transient analysis, defined in the **.TRAN** command.

- **R**

  Specifies a periodically repetitive signal of period `TN`. The signal repeats from the beginning.

repeat is the time, in units of seconds, which specifies the start point of the waveform to repeat. This time needs to be less than the greatest time point, tn.

- **R**=VAL

  Specifies a time value of the repeating function. The period will be equal to the last time value specified in the PWL statement, `TN`, minus the **R** value.

- **PWLPERIOD**=VAL

  Alternative to **R**=VAL. Specifies the period of the periodic waveform.

- **SHIFT**=VAL

  This acts as if the **SHIFT** value was added to all time values specified in the PWL card.

- **SCALE**=VAL

  Element scale factor. The value of the element is multiplied by **SCALE**, which defaults to 1.

- **STRETCH**=VAL

  Time scale factor applied to the waveform. Default value is 1.

- **FILE**=pwl_file

  Allows Eldo to read PWL corner points from the specified file *pwl_file*. This is a text file that supplies the time-current (tn in) or time-voltage (tn vn) pairs. Engineering units (for example 9ns) are allowed. The time-value pairs are separated by spaces, commas or newline characters. The file can have multiple lines and can have any number of point pairs per line. Continuation signs "+" are not needed.

- **COL**=VAL

  For use with multi-column file specification. Selects the column number of the files. The time values column is the column 0. So a value less than 1 is not allowed for col.

- **ISTART**=VAL

  For use with multi-column file specification. Selects the starting line of data. First data line is 1.

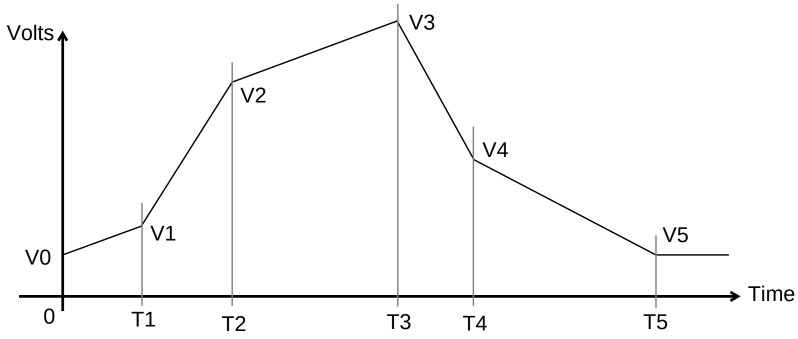
- **ISTOP**=VAL

  For use with multi-column file specification. Selects the stopping line of data.

- **ISTEP**=VAL

  For use with multi-column file specification. Selects the data interval. A value of nb means Eldo peaks data at each nb lines.

Figure 5-7. Piece Wise Linear Function

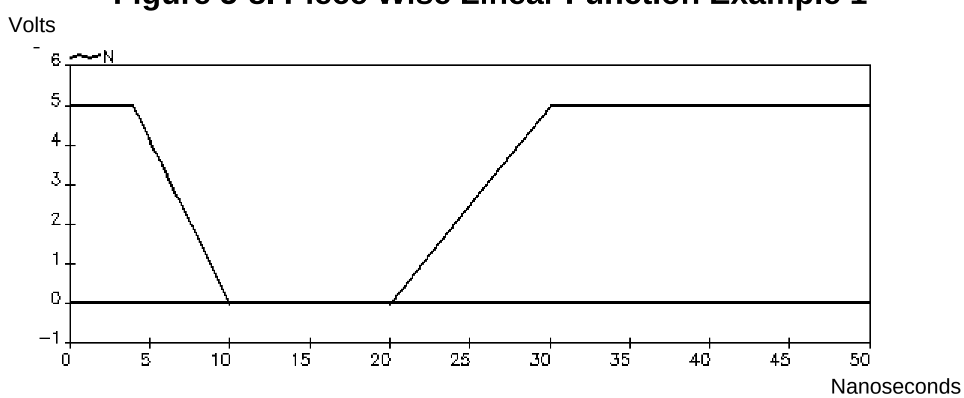


Examples

```
v1 n3 n4 pwl (4n 5 10n 0 20n 0 30n 5)
```

The voltage source v1 placed between node n3 and n4 specifies a signal which is defined by linear interpolation between the values enclosed in the parentheses.

Figure 5-8. Piece Wise Linear Function Example 1

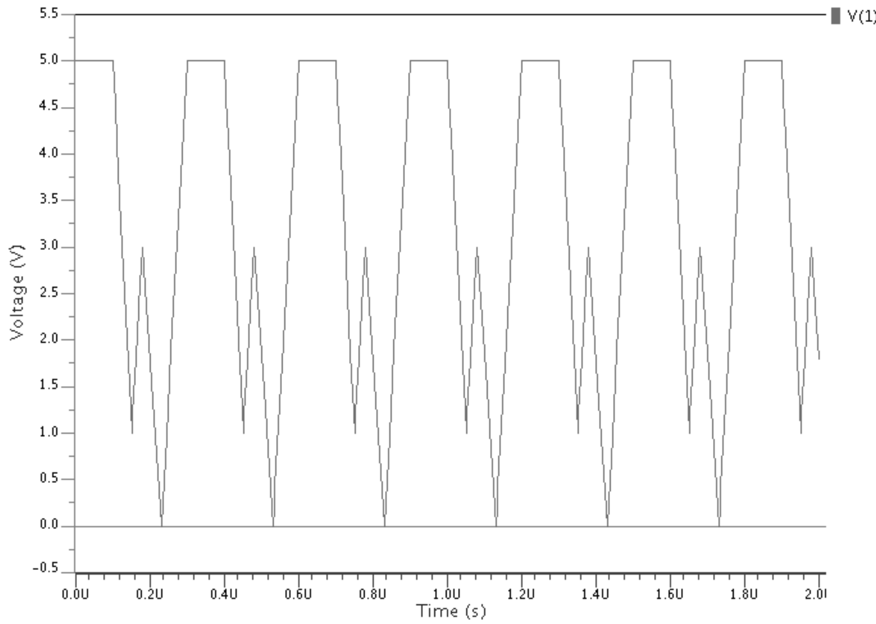


A periodically repetitive signal can be specified as shown in the following example:

```
v1 1 0 pwl (100n 5 150n 1 180n 3 230n 0 300n 5 R)
r1 1 0 1
.tran 1u 2u
.plot tran v(1)
.end
```

The voltage source v1 between nodes 1 and 0 will be interpolated between the specified values of VN at time TN and plotted until the time 2µs is reached. The repetitive part of the waveform starts at 0ns and has a period of TN=300ns. This is illustrated in Figure 5-9.

Figure 5-9. Piece Wise Linear Function Example 2



A high-impedance can be specified in PWL statements as shown in the following example:

```
V1 1 0 PWL (0 0 10n 15 15n Z 25n Z 30n 0)
```

At 15ns, v1 is disconnected which means that the rest of the circuit will impose a value on node 1, that is, node 1 becomes a node to be solved as any other node in the circuit. At 30ns, v1 is connected back.

The example below shows the usage of parameters **SCALE** and **STRETCH**:

```
v1 1 0 pwl ( 0 0 10n 10 SCALE=2)
*v1 1 0 pwl ( 0 0 10n 10 STRETCH=2)
r1 1 0 1
.tran 1n 10n
.plot tran v(1)
.end
```

When the voltage source with the PWL function is specified with the **SCALE** parameter the element is multiplied by a scale factor of 2 along the y-axis. When **STRETCH** is specified the element is multiplied by a time scale factor along the x-axis.

The example below shows two ways of specifying the **FILE** parameter. The name can be specified directly or via a parameter value.

```
v1 1 0 pwl file="stim1.txt" R

.param STIMFILE='"stim.txt"'
 v2 2 0 pwl file=$(STIMFILE)  R
```

The example below shows a PWL source with multi-column file specification.

```
*test with multicolumn files
*
v1 1 0 dc 0 pwl(file="stim4.txt" col=1 istep=1)
v2 2 0 dc 0 pwl(file="stim4.txt" col=2 istep=2)
r1 1 0 1
r2 2 0 1
.tran 1u  6u
.plot tran v(1) v(2)
.end
```

Contents of *stim4.txt* file:

```
#time v1 v2
0 1.0 9.0
500e-9 2.0 2.0
1e-6 3.0 13.0

2e-6 4.0 4.0
3e-6 5.0 15.0
4e-6 6.0 6.0
5e-6 7.0 7.0

v1 1 0 pwl file="stim1.txt" R

.param STIMFILE='"stim.txt"'
v2 2 0 pwl file=$(STIMFILE)  R
```

The file *stim4.txt* is a multi-column set of data. This file is structured in a such a way that each line consists of a time value, TN, and source values, VN1, VN2. This example is a three column file. The columns are internally labeled beginning 0 (0, 1, 2).

For the first PWL source in the main netlist:

```
v1 1 0 dc 0 pwl(file="stim4.txt" col=1 istep=1)
```

Eldo will parse the source value in column 1 with a step of 1. It is equivalent to writing:

```
v1 1 0 dc 0 pwl(0 1.0 500ns 2.0 1us 3.0 2us 4  3us 5 4us 6  5us 7)
```

For the second PWL source in the main netlist:

```
v2 2 0 dc 0 pwl(file="stim4.txt" col=2 istep=2)
```

Eldo will parse the source value in column 2 with a step of 2. It is equivalent to writing:

```
v2 2 0 dc 0 pwl(0 9.0 1us 13.0 3us 15 5us 7)
```

When the voltage source with the PWL function is specified with the **SCALE** parameter the element is multiplied by a scale factor of 2 along the y-axis. When **STRETCH** is specified the element is multiplied by a time scale factor along the x-axis.

Single Frequency FM Function

SFFM (SO SA [FC [MDI [FS]]]**)**

Generates a single frequency FM modulated signal. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Parameters

- SO

  Offset voltage in volts or current in amperes.

- SA

  Magnitude of signal in volts or amperes.

- FC

  Carrier frequency in Hertz. Default value is 1/TSTOP.

- MDI

  Modulation index. Default value is zero.

- FS

  Signal frequency in Hertz. Default value is 1/TSTOP.

The shape of the waveform is described by:

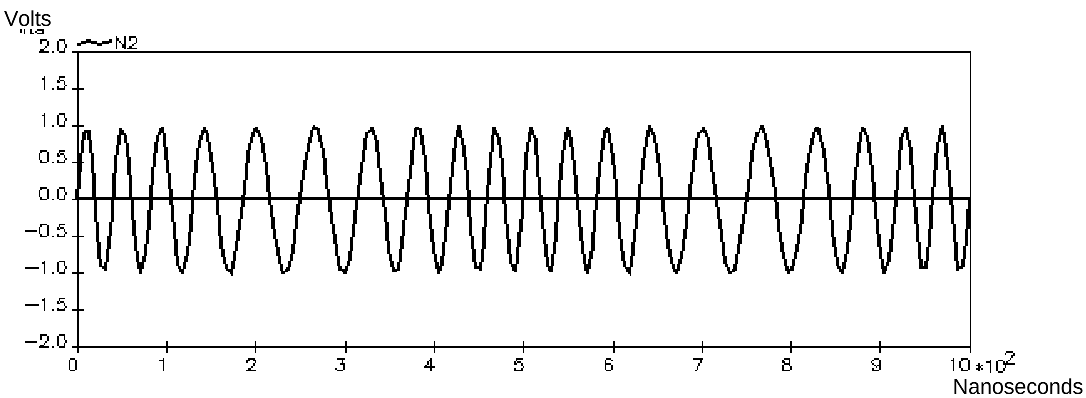
$$value = SO + SA \times \sin((2\pi \times FC \times time) + MDI \times \sin(2\pi \times FS \times time))$$

Example

```
v1 n12 0 sffm (0 1 20meg 2.5 2meg)
```

Specifies a voltage source v1 placed between node n12 and ground. The voltage has 0 offset and an amplitude of 1 V. The carrier frequency of 20 MHz is modulated with the signal frequency of 2 MHz, the modulation index being 2.5.

Figure 5-10. Single Frequency FM Function



Sine Function

```
SIN (VO VA [FR [TD [THETA [PHASE]]]])
```

Generates a sinusoidal or a damped sine wave. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Parameters

- VO

  Offset voltage in volts or offset current in amperes.

- VA

  Sine wave nominal starting amplitude in volts or amperes.

- FR

  Frequency in Hertz. Default value is 1/TSTOP.

- TD

  Delay time in seconds. Default value is zero.

- THETA

  Damping factor in 1/sec. Default value is zero.

- PHASE

  Phase delay in degrees. Default value is zero.

The waveform is described by:

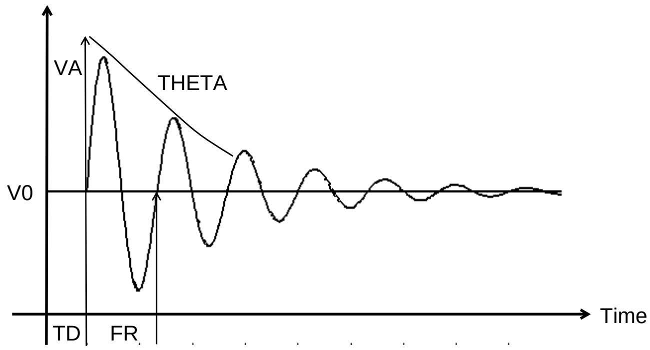
for $t < TD$

$$V = VO + VA \times \sin\left(2\pi \times \frac{PHASE}{360}\right)$$

for $t \geq TD$

$$V = VO + VA \times \exp(-(t - TD) \times THETA) \times \sin\left(2\pi \times \left(FR \cdot (t - TD) + \frac{PHASE}{360}\right)\right)$$

Figure 5-11. Sine Function

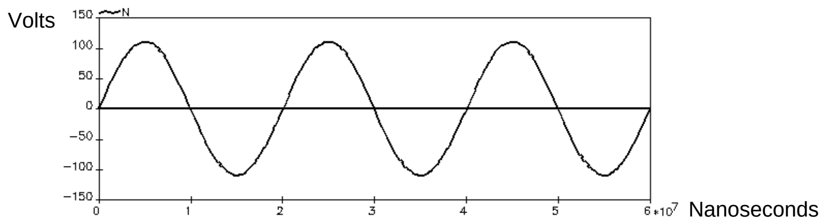


Examples

```
vsin n2 n3 sin (0 110 50 0 0)
```

Specifies the voltage source vsin between nodes n2 and n3 of amplitude 110 V and frequency 50 Hz.

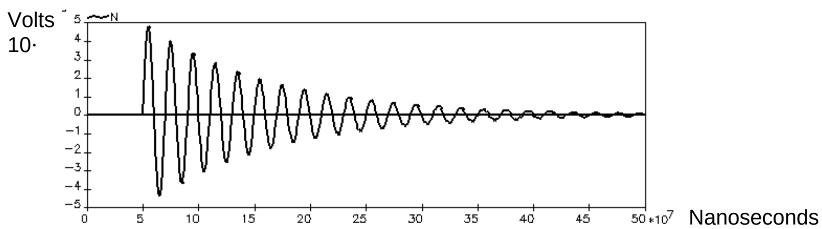
Figure 5-12. 1st Sine Function Example




```
vsin n4 n9 sin(0 50 50 .05 9)
```

Specifies the voltage source vsin placed between nodes n4 and n9 with an amplitude of 50 V and frequency of 50 Hz. It has a delay of 0.05 s with a decay factor of 9.

Figure 5-13. 2nd Sine Function Example



Trapezoidal Pulse With Bit Pattern Function

```
PBIT V0 V1 TD TD01 TR01 TD10 TF10 BITTIME {PATTERN} [R]
```

Generates a trapezoidal pulse with bit pattern source. This describes a bit pattern as a series of trapezoidal pulses with linear rise and fall waveforms. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Parameters

- V0

  Voltage (or current) representing the zero bit value.

- V1

  Voltage (or current) representing the one bit value.

- TD

  Delay before the series is started. The value assigned during this time is the first string value of the series.

- TD01

  Time delay for 0-1 bit transition in seconds. Default is 0.

- TR01

  Rise time for 0-1 bit transition in seconds. Default is TSTEP.

- TD10

  Time delay for 1-0 bit transition in seconds. Default is 0.

- TF10

  Fall time for 1-0 bit transition in seconds. Default is TSTEP.

- BITTIME

  Bit time for complete transition in seconds.

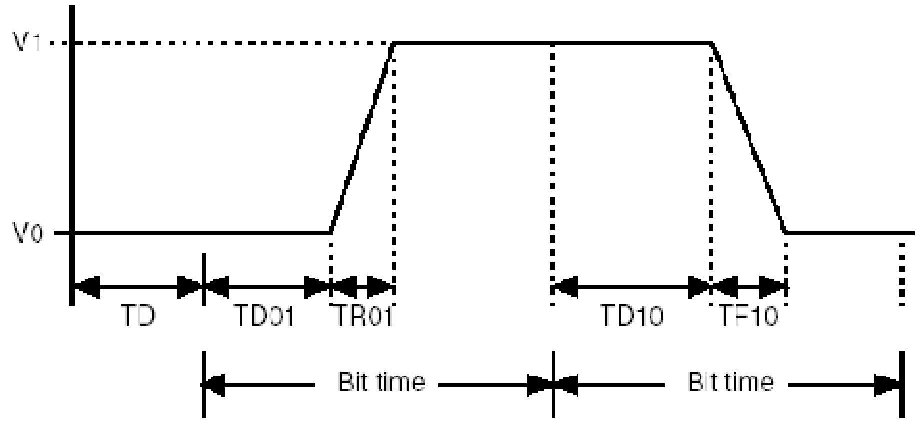
- PATTERN

  Pattern depicting value at end of each bit time.

- R

  Specifying R at the end of the pattern means that it is periodic.

Figure 5-14. Trapezoidal Pulse with Bit Pattern



Examples

```
v1 1 0 pbit 1 5 0n 1n 0.5n 0n 1n 5n 101011101000011 R
```

Specifies the voltage source, V1 as a trapezoidal voltage pulse source between nodes 1 and 0. The source has the following characteristics:

The zero bit value is 1V. The one bit value is 5V.
There is no delay before the series is started.
Time delay for 0-1 bit transition is 1ns.
Rise time for 0-1 bit transition is 0.5ns.
Time delay for 1-0 bit transition is 0ns.
Fall time for 1-0 bit transition is 1ns.
Bit time for complete transition is 5ns.

The bit pattern has fifteen bits which is repeated until the simulation run ends.

Exponential Pulse With Bit Pattern Function

```
EBIT V0 V1 TD TD01 TAU01 TD10 TAU10 BITTIME {PATTERN} [R]
```

Generates an exponential pulse with bit pattern source. This describes a bit pattern as a series of exponential pulses with exponential rise and fall waveforms. To be used in combination with independent voltage (**V**xx) or current (**I**xx) sources.

Parameters

- V0

  Voltage (or current) representing the zero bit value.

- V1

  Voltage (or current) representing the one bit value.

- TD

  Delay before the series is started. The value assigned during this time is the first string value of the series.

- TD01

  Time delay for 0-1 bit transition in seconds. Default is 0.

- TAU01

  Rise time constant for 0-1 bit transition in seconds. Default is TSTEP.

- TD10

  Time delay for 1-0 bit transition in seconds. Default is 0.

- TAU10

  Fall time constant for 1-0 bit transition in seconds. Default is TSTEP.

- BITTIME

  Bit time for complete transition in seconds.

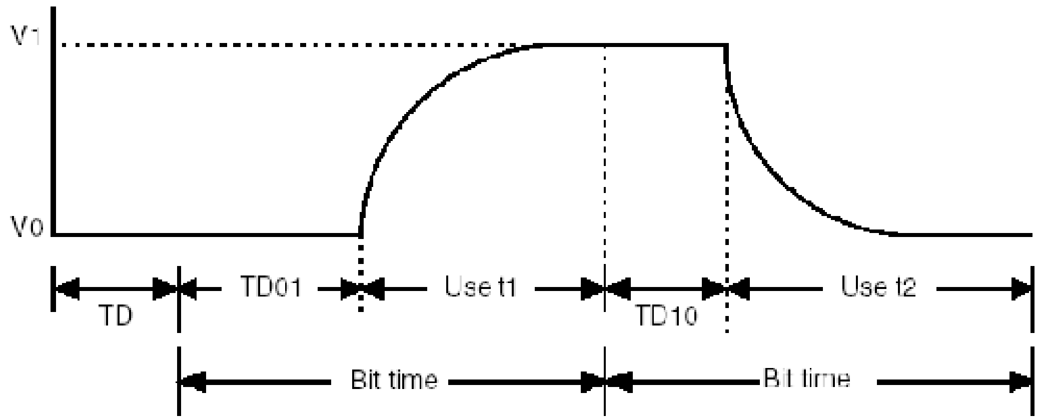
- PATTERN

  Pattern depicting value at end of each bit time.

- R

  Specifying R at the end of the pattern means that it is periodic.

Figure 5-15. Exponential Pulse with Bit Pattern



Examples

```
v1 1 0 ebit 1 5 0n 1n 0.5n 0n 1n 5n 101011101000011 R
```

Specifies the voltage source, V1 as an exponential voltage pulse source between nodes 1 and 0. The source has the following characteristics:

The zero bit value is 0V. The one bit value is 5V.
There is no delay before the series is started.
Time delay for 0-1 bit transition is 1ns.
Rise time constant for 0-1 bit transition is 0.5ns.
Time delay for 1-0 bit transition is 0ns.
Fall time constant for 1-0 bit transition is 1ns.
Bit time for complete transition is 5ns.

The bit pattern has fifteen bits which is repeated until the simulation run ends.

Dependent Sources

Voltage Controlled Voltage Source

Linear (Voltage Gain Block)

```
Exx NP NN [VCVS] NCP NCN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
Exx NP NN [VCVS] NCP NCN VAL0 {VALn} [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Polynomial

```
Exx NP NN [VCVS] POLY(ND) PCP PCN {PCP PCN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Piece Wise Linear

```
Exx NP NN PWL(1) NCP NCN PWL_LIST [DELTA=val]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Multi-Input Gate

```
Exx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) PCP PCN {PCP PCN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Delay Element

```
Exx NP NN DELAY NCP NCN [TD=val] [ABS=VAL]
```

Arithmetic Expression

```
Exx NP NN VALUE={EXPR} [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Tabular

```
Exx NP NN [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL]
+ TABLE EXPR=(XN YN) {(XN YN)} [ABS=VAL]
```

Integral/Derivative

```
Exx NP NN INTEGRATION|DERIVATION NCP NCN VAL
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

S-domain

```
Exx NP NN FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
Exx NP NN PZ  NCP NCN a zr1 zi1 ... zrm zim, b pr1 pi1 ... prn pin
```

Frequency Dependent

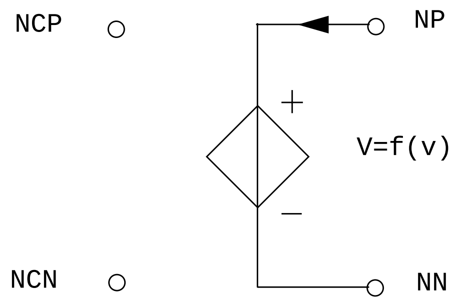
```
Exx NP NN FREQ NCP NCN f0 a0 ph0 f1 a1 ph1... fn an phn
+ [RESTORE_CAUSALITY=0|1]
```

Ideal Transformer

```
Exx NP NN TRANS[FORMER] NCP NCN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Ideal Op-Amp

```
Exx NP NN OPAMP NCP NCN
```



Note

The current flows from the positive node `NP`, through the current source, to the negative node `NN`.

Parameters

- `xx`

  Voltage controlled voltage source name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `NCP`

  Name of the positive controlling node.

- `NCN`

  Name of the negative controlling node.

- `VAL`

  Voltage gain.

The above parameters can be related by:

$$V(NP) - V(NN) = VAL \times (V(NCP) - V(NCN))$$

- `VALn`

  Coefficients of the polynomial function:

  `VAL0` is a voltage shift,
  `VAL1` is the 1st order gain,
  `VAL2` is the 2nd order gain, and so on

The above parameters can be related by:

$$V(NP, NN) = VAL0 + VAL1 \times V(NCP, NCN) + VAL2 \times V(NCP, NCN)^2 + VAL3 \times V(NCP, NCN)^3 + \ldots$$

- **MIN**=VAL

  Minimum output voltage value. Unit Volts.

- **MAX**=VAL

  Maximum output voltage value. Unit Volts.

- **TC1**, **TC2**

  First and Second order temperature coefficients. Default values are zero. The output value is updated with temperature according to the formula:

  $$VAL(T) = VAL(TNOM) \times (1 + TC1(T - TNOM) + TC2(T - TNOM)^2)$$

- **SCALE**=VAL

  Element scale factor. The value of the element is multiplied by **SCALE**, which defaults to 1.

- **ABS**=val

  Can be set to 1 or 0 (default is 0). If **ABS**=1, the output value is the absolute value of the signal.

- **POLY**

  Keyword indicating the source has a non-linear polynomial description.

- ND

  Order of the polynomial when **POLY** is specified. Number of Inputs when **NAND**, **NOR**, **AND**, **OR** is specified. ND must be greater than or equal to 1. If ND is not specified it will default to 1.

- PCP

  Name of the positive controlling node producing the voltage difference for the function arguments of the polynomial. Number is equivalent to the order of the polynomial.

- PCN

  Name of the negative controlling node producing the voltage difference for the function arguments of the polynomial. Number is equivalent to the order of the polynomial.

- PN

  Coefficients of the polynomial.

- **NAND**, **NOR**, **AND**, **OR**

  One of these can be specified in place of the existing keyword **POLY**. If **NAND** or **AND** are used, the lower valued command will be used to compute the output. In the case of **NOR** or **OR**, the higher valued command will be used. In such cases of **NAND**/**AND**/**OR**/**NOR** types, Eldo expects a list of couple values (x,y) specified as a `PWL_LIST`, the output will be the interpolated value y. Commas used as delimiters are optional.

- **`PWL`**`(1)`

  When specified, a simple interpolation (straight line between two points) will be performed to evaluate the output. A list of couple values (x,y) specified as a `PWL_LIST` is expected.

- `PWL_LIST`

  Consists of a list of couple values (x,y); interpolation will be made to extract the value, depending on `(v(ncp)-v(ncn))`. x is the voltage value across the controlling nodes `NCP` and `NCN`, y is the corresponding output value.

- **`DELTA`**`=VAL`

  This parameter must be in the range of 0 to 0.5, and is used to smooth out the output. The smooth-out occurs in the portion of the interval determined by the **`DELTA`** value. If **`DELTA`** is 0, smooth-out does not occur and strict linear interpolation is performed to compute the output. If **`DELTA`** is set to 0.5 the smooth-out occurs over the whole interval.

- **`DELAY`**

  The **`E`** element is controlled by the voltage, therefore, the item specified after the **`DELAY`** operator must be the values of the positive and negative control nodes. The output is shifted by a delay value of **`TD`**.

- **`TD`**`=VAL`

  Delay value. The default value of **`TD`** is 0 if left unspecified.

- **`VALUE`**

  Keyword indicating that the source has a functional description. A set of expressions is specified in `EXPR`.

- **`TABLE`**

  Keyword indicating that the source has a tabular description. The table itself contains pairs of values. `EXPR` is evaluated, and its value is used to look up an entry in the table. Linear interpolation is made between entries.

- `EXPR`

  A set of expressions, used to set the source value or an entry look up for a tabular description of the source.

Expression formats are described in the Eldo Control Language chapter.

Controlled source expressions may also contain voltages, currents or time. Voltages may be the voltage through a node, for example **`v(`**`6`**`)`**, or the voltage across two nodes **`v(`**`5, 6`**`)`**. Currents must be the current through a voltage source, such as **`i(`**`v1`**`)`**.

`EXPR` is also able to make use of the operators **`DDT`** and **`IDT`**. **`DDT`** stands for derivative, and **`IDT`** for integral. **`DDT`** and **`IDT`** operators utilize the integration scheme which is used by Eldo for computing the derivative/integral.

Please refer to "Arithmetic Functions" on page 78.

- `XN, YN`

  Input and corresponding output source values for tabular source definitions.

- **`INTEGRATION|DERIVATION`**

  Specifies that the voltage drop across `NP` and `NN` should be equal to the *integral* (or *derivative*) of ***v***(`NCP`)–***v***(`NCN`) multiplied by `VAL`.

- **`FNS`**

  Specifies a s-domain function for which the transfer function is:

$$H = \frac{n0 + n1 \cdot s \ldots + n_m \cdot s^n}{p0 + p1 \cdot s \ldots + p_m \cdot s^n}$$

  If ***P2*** is 0 and ***NCN*** is 0, this is equivalent to the existing **`FNS`** device:

```
FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
```

- `n0 n1 ... nm, p0 p1 ... pn`

  Polynomial coefficients for the transfer function of **`FNS`**.

- **`PZ`**

  When the poles and zeroes are known, the transfer function is:

$$H = \frac{a \cdot \Pi((s + (zri + j \cdot 2 \cdot pi \cdot zii)) \cdot (s + (zri - 2 \cdot pi \cdot zii)))}{b \cdot \Pi((s + (pri + j \cdot 2 \cdot pi \cdot pii)) \cdot (s + (pri - 2 \cdot pi \cdot pii)))}$$

  For the numerator, ***i*** is from 1 to m. For the denominator, ***i*** is from 1 to n.

Note

The conjugate appears only if the imaginary part is not 0.

- `a, b`

  Coefficients for the transfer function of **`PZ`**.

- `pr, pi`

  Poles of the transfer function. `pr` is the real part, `pi` the imaginary part.

- `zr, zi`

  Zeroes of the transfer function. `zr` is the real part, `zi` the imaginary part.

- **`FREQ`**

  Specifies a frequency domain description. Enables a frequency dependent voltage controlled source to be simulated for AC, Transient, and MODSST analyses.

- `f0 a0 ph0 f1 a1 ph1... fn an phn`

  Coefficients for the frequency domain. `fi`, `ai` and `phi` are the frequency, the amplitude in dB, and the phase in degrees respectively. At each frequency point, Eldo will evaluate the amplitude in dB by making a log interpolation, and will evaluate the phase by making a linear interpolation.

- **`RESTORE_CAUSALITY`**`=0|1`

  Having a purely real impedance that varies with the square root of frequency is not physically realizable. It would be non-causal. Thus when modeling a physical resistance, specifying a variation as the square root of frequency can be interpreted by Eldo as either:

  - specifying the module of the impedance (default, or **`RESTORE_CAUSALITY`**=0), or
  - specifying the real part of the impedance, and Eldo computes the imaginary part (**`RESTORE_CAUSALITY`**=1)

- **`TRANS`**`[`**`FORMER`**`]`

  Keyword specifying that an ideal transformer is used. It differs from a linear VCVS source by the equations used:

$$V(NP) - V(NN) = \frac{V(NCP) - V(NCN)}{VAL}$$

- **`OPAMP`**

  Keyword specifying that an ideal operational amplifier is used. This models an ideal amplifier of infinite gain. Note: **`OPAMP`** is a keyword here; use Exx outp outn POLY(1) OPAMP 0 gain if OPAMP is actually a pin name and if a regular VCVS of gain ‘gain’ is required.

Examples

```
e23 n2 n3 14 0 2.0
```

Specifies that the voltage applied between nodes n2 and n3 is twice the potential difference between node 14 and ground.

```
e1 2 0 poly (2) 3 0 5 0 0 1 1 2 4.5
```

Specifies a 2nd order non-linear voltage controlled voltage source e1 between node 2 and ground. The two controlling voltages contributing to the voltage difference for the function arguments f_a and f_b occur between node 3 (NCP1) and ground (NCN1), and node 5 (NCP2) and ground (NCN2). Polynomial coefficients are P0=0, P1=1, P2=1, P3=2 and P4=4.5. The resultant non-linear voltage function has the following form:

$$f_v = f_a + f_b + 2f_a^2 + 4.5f_af_b$$

where:

f_a is the voltage difference between the controlling nodes NCP1 and NCN1.

f_b is the voltage difference between the controlling nodes NCP2 and NCN2.

```
e1 1 2 value = {v(3,4)*i(v5)}
```

Specifies that the voltage applied between nodes 1 and 2 is the instantaneous power calculated by multiplying the voltage across nodes 3 and 4 and the current through V5.

```
V1 A 0 1
r1 A 0 1
V2 B 0 5
r2 B 0 5
E3 3 0 NAND(2) A 0 B 0 (-10,-30) (10,30)
r3 3 0 1
```

This example shows the use of the **NAND** keyword. In this case, the command of lower value is used, i.e. it is v(A,0) which will be used, the voltage on node 3 will be 3V (interpolation for x in [-10,10] and y in [-30,30]).

```
V1 A 0 1
r1 A 0 1
E3 3 0 PWL(1) A 0 (-10,-30) (10,30)
r3 3 0 1
.DC V1 1 10 1
```

Specifies a voltage controlled voltage source between nodes 3 and 0. The voltage across `E3` is controlled by the PWL voltage at node A. The **.DC** command changes the voltage at node A between 1V and 10V. With the VCVS definition, V(3) changes according to the voltage on A:

when V(A)=-10, I(G3)=–30
when V(A)=10, I(G3)=30

then by linear interpolation, we find that:

when V(A)=1, I(G3)=3,
when V(A)=10, I(G3)=30.

```
*SEARCH MAX must be 3.7276 and min must be 2.2758
E2 2 0 FREQ 1 0
+1k  20 0
+1e10 5 0
v1 1 0 2 ac 1
r1 1 2 1k
r2 2 0 1k
.ac dec 10 1e7 1e9
```

Specifies a voltage controlled voltage source in the frequency domain. It is between nodes 2 and 0 and the voltage across those two pins is equal to the GAIN(f) multiplied by V(1,0).

Current Controlled Current Source

Linear (Current Gain Block)

```
Fxx NP NN [CCCS] VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Polynomial

```
Fxx NP NN [CCCS] POLY(ND) VN {VN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL]
+ [ABS=VAL]
```

Piece Wise Linear

```
Fxx NP NN PWL(1) VN PWL_LIST [DELTA=val]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Multi-Input Gate

```
Fxx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) VN {VN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Delay Element

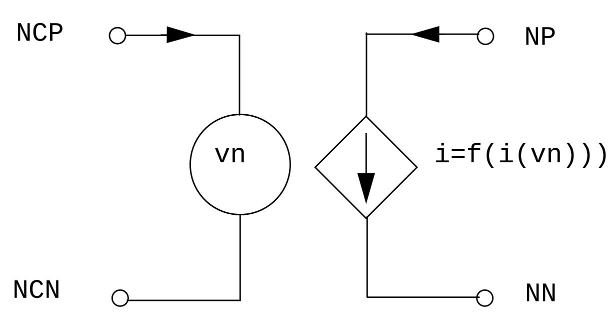
```
Fxx NP NN DELAY VN [TD=val] [ABS=VAL]
```

Integral/Derivative

```
Fxx NP NN INTEGRATION|DERIVATION VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

S-domain

```
Fxx NP NN FNS VN n0 n1 ... nm, p0 p1 ... pn
Fxx NP NN PZ VN a zr1 zi1 ... zrm zim, b pr1 pi1 ... prn pin
```



Note
The current flows from the positive node `NP`, through the current source, to the negative node `NN`. `NCP` and `NCN` are the positive and negative controlling nodes for source `vn`.

Parameters

- XX

  Current controlled current source name.

- NP

  Name of the positive node.

- NN

  Name of the negative node.

- VN

  Name of the 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 VN. When **POLY** is specified, one name must be added for each dimension of the polynomial. If **POLY** is not specified, it is assumed that there is only one dimension for which a name should be added.

- VAL

  Current gain.

The above parameters can be related by:

$$I(Fxx) = I(VNAM) \times VAL$$

- **MIN**=VAL

  Minimum output current value. Unit Amps.

- **MAX**=VAL

  Maximum output current value. Unit Amps.

- **TC1**, **TC2**

  First and Second order temperature coefficients. Default values are zero. The output value is updated with temperature according to the formula:

$$VAL(T) = VAL(TNOM) \times (1 + TC1(T - TNOM) + TC2(T - TNOM)^2)$$

- **SCALE**=VAL

  Element scale factor. The value of the element is multiplied by SCALE, which defaults to 1.

- **ABS**=val

  Can be set to 1 or 0 (default is 0). If **ABS**=1, the output value is the absolute value of the signal.

- **POLY**

  Keyword indicating the source has a non-linear polynomial description.

- `ND`

  Order of the polynomial when **POLY** is specified. Number of Inputs when **NAND**, **NOR**, **AND**, **OR** is specified. `ND` must be greater than or equal to 1. If `ND` is not specified it will default to 1.

- `PN`

  Coefficients of the polynomial.

- **NAND**, **NOR**, **AND**, **OR**

  One of these can be specified in place of the existing keyword **POLY**. If **NAND** or **AND** are used, the lower valued command will be used to compute the output. In the case of **NOR** or **OR**, the higher valued command will be used. In such cases of **NAND**/**AND**/**OR**/**NOR** types, Eldo expects a list of device names (x,y) specified as a `PWL_LIST`, the output will be the interpolated value y. Commas used as delimiters are optional.

- **PWL**(1)

  When specified, a simple interpolation (straight line between two points) will be performed to evaluate the output. A list of couple values (x,y) specified as a `PWL_LIST` is expected.

- `PWL_LIST`

  Consists of a list of couple values (x,y); interpolation will be made to extract the value, depending on `i(vn)`. x is the current through the controlling node `VN`, y is the corresponding output value.

- **DELTA**=VAL

  This parameter must be in the range of 0 to 0.5, and is used to smooth out the output. The smooth-out occurs in the portion of the interval determined by the **DELTA** value. If **DELTA** is 0, smooth-out does not occur and strict linear interpolation is performed to compute the output. If **DELTA** is set to 0.5 the smooth-out occurs over the whole interval.

- **DELAY**

  The **F** element is controlled by the current, therefore, the item specified after the **DELAY** operator must be a device name, `VN`. The output is shifted by a **DELAY** value of **TD**.

- **TD**=VAL

  Delay value. The default value of `TD` is 0 if left unspecified.

- **INTEGRATION|DERIVATION**

  Specifies that the current flowing through **F**xx should be equal to the *integral* (or *derivative*) of ***i***(VN) multiplied by `VAL`.

- **FNS**

  Specifies a s-domain function for which the transfer function is:

$$H = \frac{n0 + n1 \cdot s... + n_m \cdot s^n}{p0 + p1 \cdot s... + p_m \cdot s^n}$$

If ***P2*** is 0 and ***NCN*** is 0, this is equivalent to the existing **`FNS`** device:

```
FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
```

- `n0 n1 ... nm, p0 p1 ... pn`

  Polynomial coefficients for the transfer function of **`FNS`**.

- **`PZ`**

  When the poles and zeroes are known, the transfer function is:

$$H = \frac{a \cdot \Pi((s + (zri + j \cdot 2 \cdot pi \cdot zii)) \cdot (s + (zri - 2 \cdot pi \cdot zii)))}{b \cdot \Pi((s + (pri + j \cdot 2 \cdot pi \cdot pii)) \cdot (s + (pri - 2 \cdot pi \cdot pii)))}$$

  For the numerator, ***i*** is from 1 to m. For the denominator, ***i*** is from 1 to n.

Note

The conjugate appears only if the imaginary part is not 0.

- `a, b`

  Coefficients for the transfer function of **`PZ`**.

- `pr, pi`

  Poles of the transfer function. `pr` is the real part, `pi` the imaginary part.

- `zr, zi`

  Zeroes of the transfer function. `zr` is the real part, `zi` the imaginary part.

Examples

```
f7 n4 n6 v9 7
```

Specifies that the current through f7 flowing from node n4 to node n6 is seven times the current of v9.

```
f1 2 0 poly (2) v1 v2 0 1 1 3
```

Specifies a 2nd order non-linear current controlled current source f1 placed between node 2 and ground. The names of the zero voltage sources sensing the current for the function arguments of the polynomial are v1 and v2. Polynomial coefficients are PN0=0, PN1=1 and PN2=3. The resultant non-linear current function has the following form:

$$f_v = f_a + f_b + 3f_a^2$$

where f_a and f_b are equal to the current flowing through V1 and V2 respectively.

```
f2 3 0 poly (3) v1 v2 v3 0 1 1 3 2.5
```

Specifies a 3rd order non-linear current controlled current source f2 placed between node 3 and ground. The names of the zero voltage sources sensing the current for the function arguments of the polynomial are v1, v2 and v3. Polynomial coefficients are PN0=0, PN1=1, PN2=1, PN3=3 and PN4=2.5.

The resultant non-linear current function has the following form:

$$f_v = f_a + f_b + 3f_c + 2.5f_a^2$$

where f_a, f_b and f_c are equal to the current flowing through V1, V2 and V3 respectively.

```
V1 A 0 1
r1 A 0 1
V2 B 0 5
r2 B 0 5
F3 3 0 NAND(2) V1 V2 (-10,-30) (10,30)
r3 3 0 1
```

This example shows the use of the `NAND` keyword. In this case, the command of lower value is used, i.e. it is v(A,0) which will be used, the voltage on node 3 will be 3V (interpolation for x in [-10,10] and y in [-30,30]).

```
V1 A 0 1
r1 A 0 1
F3 3 0 PWL(1) V1 (-10,-30) (10,30)
r3 3 0 1
.DC V1 1 10 1
```

V(3) will vary from 3 to 30 when V(1) varies from 1 to 10.

Voltage Controlled Current Source

Linear (Transconductance Gain Block)

```
Gxx NP NN [VCR|VCCAP|VCCS] NCP NCN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Polynomial

```
Gxx NP NN [VCR|VCCAP|VCCS] POLY(ND) PCP PCN {PCP PCN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Piece Wise Linear

```
Gxx NP NN [VCR|VCCAP|VCCS] [PWL(1)|NPWL(1)|PPWL(1)] NCP NCN PWL_LIST
+ [DELTA=val] [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL]
+ [ABS=VAL]
```

Multi-Input Gate

```
Gxx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) PCP PCN {PCP PCN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Delay Element

```
Gxx NP NN DELAY NCP NCN [TD=val] [ABS=VAL]
```

Arithmetic Expression

```
Gxx NP NN VALUE={EXPR} [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Tabular

```
Gxx NP NN [VCR|VCCAP|VCCS] [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL]
+ [SCALE=VAL] [ABS=VAL] TABLE EXPR=(XN YN) {(XN YN)}
```

Integral/Derivative

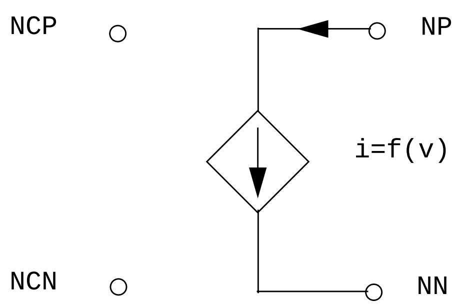
```
Gxx NP NN INTEGRATION|DERIVATION NCP NCN VAL
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

S-domain

```
Gxx NP NN FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
Gxx NP NN PZ  NCP NCN a zr1 zi1 ... zrm zim, b pr1 pi1 ... prn pin
```

Frequency Dependent

```
Gxx NP NN FREQ NCP NCN f0 a0 ph0 f1 a1 ph1... fn an phn
+ [RESTORE_CAUSALITY=0|1]
```



Note

The current flows from the positive node `NP`, through the current source, to the negative node `NN`.

Parameters

- `xx`

  Voltage controlled current source name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `NCP`

  Name of the positive controlling node.

- `NCN`

  Name of negative controlling node.

- `VAL`

  Transadmittance in Ω^{-1}.

The above parameters can be related by:

$$I(Gxx) = VAL(V(NCP) - V(NCN))$$

- **`MIN`**`=VAL`

  Minimum output current value. Unit Amps.

- **`MAX`**`=VAL`

  Maximum output current value. Unit Amps.

- **TC1**, **TC2**

  First and Second order temperature coefficients. Default values are zero. The output value is updated with temperature according to the formula:

  $$VAL(T) = VAL(TNOM) \times (1 + TC1(T - TNOM) + TC2(T - TNOM)^2)$$

- **SCALE**=VAL

  Element scale factor. The value of the element is multiplied by **SCALE**, which defaults to 1.

- **ABS**=val

  Can be set to 1 or 0 (default is 0). If **ABS**=1, the output value is the absolute value of the signal.

- **POLY**

  Keyword indicating the source has a non-linear polynomial description.

- ND

  Order of the polynomial when **POLY** is specified. Number of Inputs when **NAND**, **NOR**, **AND**, **OR** is specified. ND must be greater than or equal to 1. If ND is not specified it will default to 1.

- PCP

  Name of the positive controlling node giving the voltage difference for the function arguments of the polynomial. Number is equal to the order of the polynomial.

- PCN

  Name of the negative controlling node giving the voltage difference for the function arguments of the polynomial. Number is equal to the order of the polynomial.

- PN

  Coefficients of the polynomial.

- **VCCAP**

  Used to instantiate a Voltage Controlled CAPacitor. The value of C is computed from the controlling nodes and polynomial coefficients in the same way that the **POLY** voltage control source values are computed.

- **VCR**

  Used to instantiate a Voltage Controlled Resistor. The value of R is computed from the controlling nodes and polynomial coefficients in the same way that the **POLY** voltage control source values are computed. VCR output has an additional formulation, the **PWL**:

- **PWL**(1)

  When specified, a simple interpolation (straight line between two points) will be performed to evaluate the output. A list of couple values (x,y) specified as a PWL_LIST is expected.

- PWL_LIST

  Consists of a list of couple values (x,y); interpolation will be made to extract the value, depending on (v(ncp)-v(ncn)). x is the voltage value across the controlling nodes NCP and NCN, y is the corresponding output value.

- **NPWL**(1)

  When specified, the command value used to evaluate the output will be: V(ncp)-V(ncn) if V(NP)>V(NN), otherwise the command value will be v(ncp)-v(NP).

- **PPWL**(1)

  When specified, the command value used to evaluate the output will be: V(ncp)-V(ncn) if V(NP)<V(NN), otherwise the command value will be v(ncp)-v(NP).

Note

For **NPWL**(1), node NCN and NN must be the same node.
For **PPWL**(1), node NCN and NP must be the same node.
Also note that if VCR values are limited to [-1.0e-15 1.0e-15] for instance, then a zero value for VCR is excluded.

- **NAND**, **NOR**, **AND**, **OR**

  One of these can be specified in place of the existing keyword **POLY**. If **NAND** or **AND** are used, the lower valued command will be used to compute the output. In the case of **NOR** or **OR**, the higher valued command will be used. In such cases of **NAND**/**AND**/**OR**/**NOR** types, Eldo expects a list of couple values (x,y) specified as a PWL_LIST, the output will be the interpolated value y. Commas used as delimiters are optional.

- **DELTA**=VAL

  This parameter must be in the range of 0 to 0.5, and is used to smooth out the output. The smooth-out occurs in the portion of the interval determined by the **DELTA** value. If **DELTA** is 0, smooth-out does not occur and strict linear interpolation is performed to compute the output. If **DELTA** is set to 0.5 the smooth-out occurs over the whole interval.

- **DELAY**

  The **G** element is controlled by the voltage, therefore, the item specified after the **DELAY** operator must be the values of the positive and negative control nodes. The output is shifted by a delay value of **TD**.

- **TD**=VAL

  Delay value. The default value of TD is 0 if left unspecified.

- **VALUE**

  Keyword indicating that the source has a functional description. A set of expressions is specified in EXPR.

- **TABLE**

  Keyword indicating that the source has a tabular description. The table itself contains pairs of values. `EXPR` is evaluated, and its value is used to look up an entry in the table. Linear interpolation is made between entries.

- `EXPR`

  A set of expressions, used to set the source value or an entry look up for a tabular description of the source.

> Expression formats are described in the Eldo Control Language chapter.

Controlled source expressions may also contain voltages, currents or time. Voltages may be the voltage through a node, for example **v(**6**)**, or the voltage across two nodes **v(**5, 6**)**. Currents must be the current through a voltage source, such as **i(**v1**)**.

`EXPR` is able to make use of the operators **DDT** and **IDT**. **DDT** stands for derivative, and **IDT** for integral. **DDT** and **IDT** operators utilize the integration scheme which is used by Eldo for computing the derivative/integral.

> Please refer to "Arithmetic Functions" on page 78.

- `XN, YN`

  Input and corresponding output source values for tabular source definitions.

- **INTEGRATION|DERIVATION**

  Specifies that the current flowing through **G**xx should be equal to the *integral* (or *derivative*) of ***v***(NCP)–***v***(NCN) multiplied by `VAL`.

- **FNS**

  Specifies a s-domain function for which the transfer function is:

$$H = \frac{n0 + n1 \cdot s ... + n_m \cdot s^n}{p0 + p1 \cdot s ... + p_m \cdot s^n}$$

If ***P2*** is 0 and ***NCN*** is 0, this is equivalent to the existing **FNS** device:

```
FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
```

- `n0 n1 ... nm, p0 p1 ... pn`

  Polynomial coefficients for the transfer function of **FNS**.

- **PZ**

  When the poles and zeroes are known, the transfer function is:

$$H = \frac{a \cdot \Pi((s-(zri + j \cdot 2 \cdot pi \cdot zii)) \cdot (s-(zri - 2 \cdot pi \cdot zii)))}{b \cdot \Pi((s-(pri + j \cdot 2 \cdot pi \cdot pii)) \cdot (s-(pri - 2 \cdot pi \cdot pii)))}$$

  For the numerator, ***i*** is from 1 to m. For the denominator, ***i*** is from 1 to n.

Note

The conjugate appears only if the imaginary part is not 0.

- `a, b`

  Coefficients for the transfer function of **`PZ`**.

- `pr, pi`

  Poles of the transfer function. `pr` is the real part, `pi` the imaginary part.

- `zr, zi`

  Zeroes of the transfer function. `zr` is the real part, `zi` the imaginary part.

- **FREQ**

  Specifies a frequency domain description. Enables a frequency dependent voltage controlled source to be simulated for AC, Transient, and MODSST analyses.

- `f0 a0 ph0 f1 a1 ph1... fn an phn`

  Coefficients for the frequency domain. `fi`, `ai` and `phi` are the frequency, the amplitude in dB, and the phase in degrees respectively. At each frequency point, Eldo will evaluate the amplitude in dB by making a log interpolation, and will evaluate the phase by making a linear interpolation.

- **`RESTORE_CAUSALITY`**`=0|1`

  Having a purely real impedance that varies with the square root of frequency is not physically realizable. It would be non-causal. Thus when modeling a physical resistance, specifying a variation as the square root of frequency can be interpreted by Eldo as either:

  - specifying the module of the impedance (default, or **`RESTORE_CAUSALITY`**=0), or
  - specifying the real part of the impedance, and Eldo computes the imaginary part (**`RESTORE_CAUSALITY`**=1)

Examples

```
g2 n2 n0 5 0 0.0005
```

Specifies that the current through g2 from node n2 to ground is equal to 0.0005 times the potential difference between node 5 and ground.

```
g1 3 0 poly (3) 3 0 9 0 5 0 0 1 1 3 2 1
```

Specifies a 3rd order non-linear voltage controlled current source g1 placed between node 3 and ground. The three controlling voltages giving the voltage difference for the function arguments occur between node 3 (NCP1) and ground (NCN1), node 9 (NCP2) and ground (NCN2), and node 5 (NCP3) and ground (NCN3). Polynomial coefficients are P0=0, P1=1, P2=1, P3=3, P4=2 and P5=1.

The resultant non-linear voltage function has the following form:

$$f_v = f_a + f_b + 3f_c + 2f_a^2 + f_a f_b$$

where:

f_a is the voltage difference between the controlling nodes NCP1 and NCN1.
f_b is the voltage difference between the controlling nodes NCP2 and NCN2.
f_c is the voltage difference between the controlling nodes NCP3 and NCN3.

```
g1 1 2 value={v(6)*v(7)}
```

Specifies current through g1 from node 1 to node 2 to be equal to the product of voltages at nodes 6 and 7.

```
g3 5 0 table v(5)=(0,0) (0.02,2.6e-3) (0.04,4.4e-3)
+ (0.06,4.2e-3)(0.08,3.7e-3) (0.10,3.5e-3)(0.12,3.9e-3)
```

Specifies a voltage controlled current source between nodes 5 and 0. Current out of node 5 is controlled via the table by the voltage at the node 5. Thus, values in the table describe the I-V device characteristics.

```
V1 A 0 1
r1 A 0 1
V2 B 0 5
r2 B 0 5
G3 3 0 NAND(2) A 0 B 0 (-10,-30) (10,30)
r3 3 0 1
```

This example shows the use of the **NAND** keyword. In this case, the command of lower value is used, i.e. it is v(A,0) which will be used, the voltage on node 3 will be 3V (interpolation for x in [-10,10] and y in [-30,30]).

```
V1 A 0 1
r1 A 0 1
G3 3 0 PWL(1) A 0 (-10,-30) (10,30)
r3 3 0 1
.DC V1 1 10 1
```

Specifies a voltage controlled current source between nodes 3 and 0. The current through **G**3 is controlled by the PWL voltage at node A. The **.DC** command changes the voltage at node A between 1V and 10V. With the VCCS definition, V(3) changes according to the voltage on A:

when V(A)=-10, I(G3)=–30
when V(A)=10, I(G3)=30

then by linear interpolation, we find that:

when V(A)=1, I(G3)=3,
when V(A)=10, I(G3)=30.

Current Controlled Voltage Source

Linear (Transresistance Gain Block)

```
Hxx NP NN [CCVS] VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Polynomial

```
Hxx NP NN [CCVS] POLY(ND) VN {VN} PN {PN}
+ [MIN=VAL] [MAX=VAL] [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Piece Wise Linear

```
Hxx NP NN PWL(1) VN PWL_LIST [DELTA=val]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Multi-Input Gate

```
Hxx NP NN NAND(ND)|AND(ND)|OR(ND)|NOR(ND) VN {VN}
+ PWL_LIST [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

Delay Element

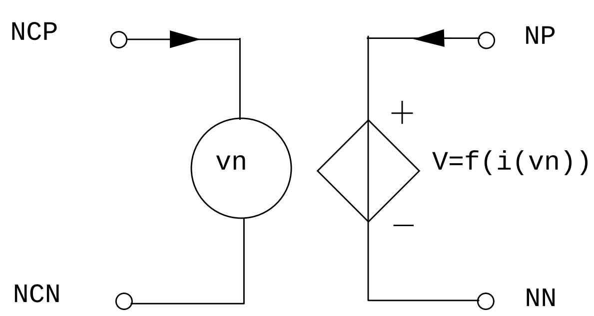
```
Hxx NP NN DELAY VN [TD=val] [ABS=VAL]
```

Integral/Derivation

```
Hxx NP NN INTEGRATION|DERIVATION VN VAL [MIN=VAL] [MAX=VAL]
+ [TC1=VAL] [TC2=VAL] [SCALE=VAL] [ABS=VAL]
```

S-domain

```
Hxx NP NN FNS VN n0 n1 ... nm, p0 p1 ... pn
Hxx NP NN PZ VN a zr1 zi1 ... zrm zim, b pr1 pi1 ... prn pin
```



Note

The current flows from the positive node `NP`, through the current source, to the negative node `NN`. `NCP` and `NCN` are the positive and negative controlling nodes for source `vn`.

Parameters

- `xx`

  Current controlled voltage source name.

- `NP`

  Name of the positive node.

- `NN`

  Name of the negative node.

- `VN`

  Name of the 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 `VN`. When **`POLY`** is specified, one name must be added for each dimension of the polynomial. If **`POLY`** is not specified, it is assumed that there is only one dimension for which a name should be added.

- `VAL`

  Transresistance in Ω.

The above parameters can be related by:

$$V(NP) - V(NN) = I(VNAM) \times VAL$$

- **`MIN`**`=VAL`

  Minimum output voltage value. Unit Volts.

- **`MAX`**`=VAL`

  Maximum output voltage value. Unit Volts.

- **`TC1`**, **`TC2`**

  First and Second order temperature coefficients. Default values are zero. The output value is updated with temperature according to the formula:

$$VAL(T) = VAL(TNOM) \times (1 + TC1(T - TNOM) + TC2(T - TNOM)^2)$$

- **`SCALE`**`=VAL`

  Element scale factor. The value of the element is multiplied by **`SCALE`**, which defaults to 1.

- **`ABS`**`=val`

  Can be set to 1 or 0 (default is 0). If **`ABS`**=1, the output value is the absolute value of the signal.

- **`POLY`**

  Keyword indicating the source has a non-linear polynomial description.

- `ND`

  Order of the polynomial when **`POLY`** is specified. Number of Inputs when **`NAND`**, **`NOR`**, **`AND`**, **`OR`** is specified. `ND` must be greater than or equal to 1. If `ND` is not specified it will default to 1.

- `PN`

  Coefficients of the polynomial.

- **NAND**, **NOR**, **AND**, **OR**

  One of these can be specified in place of the existing keyword **POLY**. If **NAND** or **AND** are used, the lower valued command will be used to compute the output. In the case of **NOR** or **OR**, the higher valued command will be used. In such cases of **NAND**/**AND**/**OR**/**NOR** types, Eldo expects a list of device names (x,y) specified as a `PWL_LIST`, the output will be the interpolated value y. Commas used as delimiters are optional.

- **PWL**(1)

  When specified, a simple interpolation (straight line between two points) will be performed to evaluate the output. A list of couple values (x,y) specified as a `PWL_LIST` is expected.

- `PWL_LIST`

  Consists of a list of couple values (x,y); interpolation will be made to extract the value, depending on `i(vn)`. x is the current through the controlling node `VN`, y is the corresponding output value.

- **DELTA**=VAL

  This parameter must be in the range of 0 to 0.5, and is used to smooth out the output. The smooth-out occurs in the portion of the interval determined by the **DELTA** value. If **DELTA** is 0, smooth-out does not occur and strict linear interpolation is performed to compute the output. If **DELTA** is set to 0.5 the smooth-out occurs over the whole interval.

- **DELAY**

  The **H** element is controlled by the current, therefore, the item specified after the **DELAY** operator must be a device name `VN`. The output is shifted by a **DELAY** value of `TD`.

- **TD**=VAL

  Delay value. The default value of `TD` is 0 if left unspecified.

- **INTEGRATION|DERIVATION**

  Specifies that the current flowing across `NP` and `NN` should be equal to the *integral* (or *derivative*) of ***i***(`VN`) multiplied by `VAL`.

- **FNS**

  Specifies a s-domain function for which the transfer function is:

$$H = \frac{n0 + n1 \cdot s ... + n_m \cdot s^n}{p0 + p1 \cdot s ... + p_m \cdot s^n}$$

  If ***P2*** is 0 and ***NCN*** is 0, this is equivalent to the existing **FNS** device:

```
FNS NCP NCN n0 n1 ... nm, p0 p1 ... pn
```

- `n0 n1 ... nm, p0 p1 ... pn`

  Polynomial coefficients for the transfer function of **`FNS`**.

- **`PZ`**

  When the poles and zeroes are known, the transfer function is:

$$H = \frac{a \cdot \Pi((s + (zri + j \cdot 2 \cdot pi \cdot zii)) \cdot (s + (zri - 2 \cdot pi \cdot zii)))}{b \cdot \Pi((s + (pri + j \cdot 2 \cdot pi \cdot pii)) \cdot (s + (pri - 2 \cdot pi \cdot pii)))}$$

  For the numerator, ***i*** is from 1 to m. For the denominator, ***i*** is from 1 to n.

Note

The conjugate appears only if the imaginary part is not 0.

- `a, b`

  Coefficients for the transfer function of **`PZ`**.

- `pr, pi`

  Poles of the transfer function. `pr` is the real part, `pi` the imaginary part.

- `zr, zi`

  Zeroes of the transfer function. `zr` is the real part, `zi` the imaginary part.

Examples

```
h7 n4 n8 v12 7.5
```

Specifies the voltage applied between nodes n4 and N8 to be equal to the current through V12 multiplied by 7.5Ω.

```
h1 2 0 poly(1) v1 1 2 4
```

Specifies a 1st order non-linear current controlled voltage source h1 placed between node 2 and ground. The name of the zero voltage source sensing the current for the function arguments is V1. Polynomial coefficients are P0=1, P1=2 and P2=4.

The resultant non-linear voltage function has the following form:

$$f_v = 1 + 2f_a + 4f_a^2$$

where f_a is equal to the current flowing through V1.

```
h2 3 0 poly(3) v1 v2 v3 0 1 1 1
```

Specifies a 3rd order non-linear current controlled voltage source H2 placed between node 3 and ground. The names of the zero voltage sources sensing the current for the function arguments are V1, V2 and V3. Polynomial coefficients are P0=0, P1=1, P2=1 and P3=1. The resultant non-linear current function has the following form:

$$f_v = f_a + f_b + f_c$$

where f_a, f_b and f_c are equal to the current flowing through V1, V2 and V3 respectively.

```
V1 A 0 1
r1 A 0 1
V2 B 0 5
r2 B 0 5
H3 3 0 NAND(2) V1 V2 (-10,-30) (10,30)
r3 3 0 1
```

This example shows the use of the **NAND** keyword. In this case, the command of lower value is used, i.e. it is v(A,0) which will be used, the voltage on node 3 will be 3V (interpolation for x in [-10,10] and y in [-30,30]).

```
V1 A 0 1
r1 A 0 1
H3 3 0 PWL(1) V1 (-10,-30) (10,30)
r3 3 0 1
.DC V1 1 10 1
```

V(3) will vary from 3 to 30 when V(1) varies from 1 to 10.

S, Y, Z Parameter Extraction

S, Y, Z Parameter Extraction

```
Vyy NP NN IPORT=VAL [RPORT=VAL] [CPORT=VAL] [LPORT=VAL] [MODE=KEYWORD]
Vyy NP NN IPORT=VAL ZPORT_FILE=string [CPORT=VAL] [LPORT=VAL]
+ [MODE=KEYWORD]
Iyy NP NN IPORT=VAL [RPORT=VAL] [CPORT=VAL] [LPORT=VAL] [MODE=KEYWORD]
Iyy NP NN IPORT=VAL ZPORT_FILE=string [CPORT=VAL] [LPORT=VAL]
+ [MODE=KEYWORD]
```

Parameters

- `yy`

  Name of the port.

- `NP`

  Name of the positive Node.

- `NN`

  Name of the Negative Node.

- **`IPORT`**

  This is a strictly positive number that is unique and is used as the port number: this number is used for naming the outputs (for instance, `.PLOT AC S(1,2)`). An error message will be issued if two port instances have the same value for **`IPORT`**, or if an **`IPORT`** is missing (for example maximum **`IPORT`** number found in the netlist is 4, and there is no instance with **`IPORT`** 3).

Optional Parameters

- **`RPORT`**

  Value of the Reference Impedance in Ohms. Default value is 50Ω.

- **`CPORT`**

  Capacitor placed in series (for V source) or parallel (for I source) with **`RPORT`**. Defaults to 0, in which case it behaves like a zero voltage source (i.e. **`CPORT`** would have no effect).

- **`LPORT`**

  Inductor placed in series (for V source) or parallel (for I source) with **`RPORT`**. Defaults to 0.

- **`ZPORT_FILE`**

  Specifies the Touchstone file name that contains the port source with a complex impedance from which the S parameters will be extracted from.

- **`MODE`**=**`SINGLE|COMMON|DIFFERENTIAL`**

  Mixed-mode S parameter selection.
  `SINGLE` specifies the port as single ended, it is dedicated to S parameter extraction. Default.
  `COMMON` and **`DIFFERENTIAL`** specify that the port is not single ended. Such ports are split

into two linked sources that are either common (same amplitude and same phase) or differential (same amplitude but opposite phases). During S parameter extraction a "non single ended" port is equally common and differential depending on which display is required. During simulation (DC, AC or TRAN) this port is either common or differential depending on the specified mode keyword.

Note

Port numbers in **V**yy instances should range from 1 to the total number of ports without discontinuity. The simulation parameters **FMIN**, **FMAX**, and Number of frequency points for the analysis are specified with a **.AC** command.

For further information, please see "Working with S, Y, Z Parameters" on page 1041.

Chapter 6
Analog Macromodels

Eldo Analog Macromodels

The following analog macromodels are provided in Eldo:

Comparator (Single Output)	COMP
Comparator (Differential Output)	COMPD
Op-amp (Linear) (Single Output)	OPAMP0
Op-amp (Linear) (Differential Output)	OPAMP0D
Op-amp (Linear 1-pole) (Single Output)	OPAMP1
Op-amp (Linear 1-pole) (Differential Output)	OPAMP1D
Op-amp (Linear 2-pole) (Single Output)	OPAMP2
Op-amp (Linear 2-pole) (Differential Output)	OPAMP2D
Delay	DEL
Saturating Resistor	SATR
Voltage Limiter	SATV
Current Limiter	SATI
Voltage Controlled Switch	VSWITCH
Current Controlled Switch	CSWITCH
Ideal Single-Pole Multiple-Throw Switch	IDEAL_SW
Triangular to Sine Wave Converter	TRI2SIN
Staircase Waveform Generator	STAIRGEN
Sawtooth Waveform Generator	SAWGEN
Triangular Waveform Generator	TRIGEN
Amplitude Modulator	AMM
Pulse Amplitude Modulator	PAM
Sample and Hold	SA_HO
Track and Hold	TR_HO
Pulse Width Modulator	PWM

Voltage Controlled Oscillator	VCO
Peak Detector	PEAK_D
Level Detector (Single & Differential Output)	LEV_D
Logarithmic Amplifier	LOGAMP
Anti-logarithmic Amplifier	EXPAMP
Differentiator	DIFF
Integrator	INTEG
Adder, Subtractor, Multiplier and Divider	ADD, SUB, MULT, DIV

General Notes on the Use of FAS Macromodels

For FAS macromodels, i.e. all those using the **Y** prefix, parameters can be declared in a number of ways. These are listed below in order of priority.

Note

Note 1
In the instantiation line, using the **PARAM** keyword. The **PARAM:** command has to be followed by a white space.
Note 2
In a model command line, the syntax is as follows:

```
.model mname modfas par=val [par=val]
```

If this syntax is used, a model name must be declared using the **MODEL:** keyword in the instantiation.
Note 3
If parameters are not explicitly declared, the parameter default values are used.

For further information on FAS models, please refer to the *CFAS User's Manual*.

Comparator

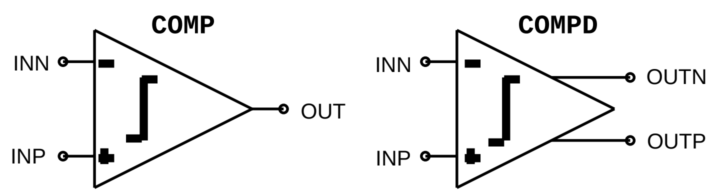
Single Output

```
COMPxx INP INN OUT [MNAME] [VHI=VAL1] [VLO=VAL2]
+ [VOFF=VAL3] [VDEF=VAL4] [TCOM=VAL5] [TPD=VAL6]
```

Differential Output

```
COMPDxx INP INN OUTP OUTN [MNAME] [VHI=VAL1] [VLO=VAL2]
+ [VOFF=VAL3] [VDEF=VAL4] [TCOM=VAL5] [TPD=VAL6]
```

Figure 6-1. Comparator Macromodel



Eldo offers two comparator macromodels, the single output comparator **COMP** and the differential output comparator **COMPD**.

Model Pins

- INP

  Name of the positive input node
- INN

  Name of the negative input node
- OUT

  Output node for the single output comparator
- OUTP

  Positive output node for the differential output comparator
- OUTN

  Negative output node for the differential output comparator
- MNAME

  Name of a model described with the **MODEL ... LOGIC** command
- **VHI**=VAL1

  Upper voltage level. Default value is 5V.
- **VLO**=VAL2

  Lower voltage level. Default value is 0V.

- **VOFF**=VAL3

  Offset input voltage. The voltage on the positive input node is compared with that on the negative added to **VOFF**. Default value is 0V.

- **VDEF**=VAL4

  Hysteresis voltage in volts. Default is 0V.

- **TCOM**=VAL5

  Commutation time. This is the time for the output to switch from **VHI** to **VLO** or vice versa. Default value is 1ns.

- **TPD**=VAL6

  Transit time through the comparator. Default value is zero.

The above parameters can be related in the following way:

Table 6-1. Comparator Parameters

VIN-	Voltage on `INN`
VIN+	Voltage on `INP`
VOUT-	Voltage on `OUT (OUTN)`
VOUT+	Voltage on `OUTP`

If $\mathrm{VIN+}(t) > \mathrm{VIN-}(t) + VOFF + VDEF$

and $\mathrm{VIN+}(t-1) < \mathrm{VIN-}(t-1) + VOFF + VDEF$

then the output rises.

If $\mathrm{VIN+}(t) < \mathrm{VIN-}(t) + VOFF - VDEF$

and $\mathrm{VIN+}(t-1) > \mathrm{VIN-}(t-1) + VOFF - VDEF$

then the output falls.

For a differential comparator

$$\mathrm{VOUT-} = (VHI + VLO) - \mathrm{VOUT+}$$

Examples

```
compd2 vp vn voutp voutn voff=0.9 vdef=0.1
+ vhi=2.5 vlo=-2.5
```

Specifies a differential output comparator compd2 with input nodes vp (+ve), vn (-ve) and output nodes voutp (+ve), voutn (-ve). The upper and lower thresholds of the comparator are +2.5V and -2.5V respectively and input offset is 0.9V. The comparator exhibits a hysteresis of 0.1V.

```
*.MODEL LOGIC definition
.model compar logic vhi=2.5 vlo=-2.5
...
comp1 vinp vinn vout compar voff=0.9 vdef=0.1
```

Specifies a single output comparator comp1 of model type compar with input nodes vinp (+ve), vinn (-ve) and output node vout. The comparator input offset voltage and hysteresis are 0.9V and 0.1V respectively. The comparator thresholds are $\pm$ 2.5V.

Op-amp (Linear)

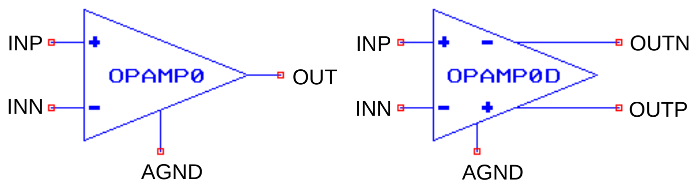
Single Output

```
Yxx OPAMP0 [PIN:] INP INN OUT AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Differential Output

```
Yxx OPAMP0D [PIN:] INP INN OUTN OUTP AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-2. Op-amp (Linear) Macromodel



Two general linear operational amplifier macromodels are available, the single output linear gain op-amp **`OPAMP0`**, and the differential output linear gain op-amp **`OPAMP0D`**. Voltage clipping effects can be described using the voltage limiter macromodel (**`SATV`**).

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUT`

  Output node for the single output op-amp
- `OUTP`

  Positive output node for the differential output op-amp
- `OUTN`

  Negative output node for the differential output op-amp
- `AGND`

  Name of the ground node

Table 6-2. Op-amp (Linear) Model Parameters

Nr.	Name	Default	Units	Definition
1	GAIN[1]	1.0×10^5		Amplifier gain
2	RIN	1.0×10^7	Ω	Input resistance

Table 6-2. Op-amp (Linear) Model Parameters

Nr.	Name	Default	Units	Definition
3	M[2]	1		Device multiplier

1. Can also be specified in dB.
2. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
yopa1 opamp0 n2 n1 n3 0 param: gain=1000 rin=10meg
```

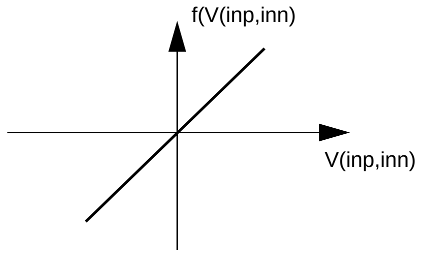
Specifies a linear gain operational amplifier yopa1 of type opamp0 having input nodes n2 (+ve) and n1 (-ve) with output node n3. Gain and input resistance parameters are declared explicitly in the instantiation.

```
.model tdk modfas gain=2000 rin=20 meg
...
yopa1 opamp0d n2 n1 n3 n4 0 model: tdk
```

Specifies a differential linear gain operational amplifier yopa1 of type opamp0d with input nodes n2 (+ve) and n1 (-ve) and output nodes n3 (-ve) and `n4` (+ve). Gain and input resistance parameters are declared using the **.MODEL** command.

Model Equations

Figure 6-3. Op-amp (Linear) Model Characteristics



General

$$I(inp) = \frac{V(inp,inn)}{RIN}$$

$$I(inn) = -\frac{V(inp,inn)}{RIN}$$

Single Output

$$V(out,agnd) = GAIN \times V(inp,inn)$$

Differential Output

$$V(outn,agnd) = -\frac{1}{2}GAIN \times V(inp,inn)$$

$$V(outp,agnd) = \frac{1}{2}GAIN \times V(inp,inn)$$

Op-amp (Linear 1-pole)

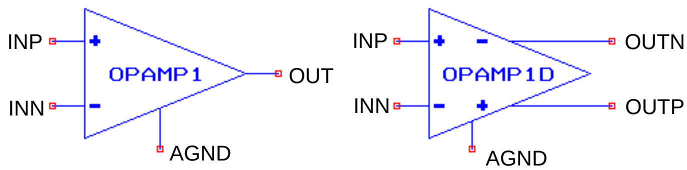
Single Output

```
Yxx OPAMP1 [PIN:] INP INN OUT AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Differential Output

```
Yxx OPAMP1D [PIN:] INP INN OUTN OUTP AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-4. Op-amp (Linear 1-pole) Macromodel



Two operational amplifier macromodels are implemented, namely the single output linear gain one pole op-amp **OPAMP1** and the differential output one pole op-amp **OPAMP1D**.

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUT`

  Output node for the single op-amp
- `OUTP`

  Positive output node for the differential output op-amp
- `OUTN`

  Negative output node for the differential output op-amp
- `AGND`

  Name of the ground node

Table 6-3. Op-amp (Linear 1-pole) Model Parameters

Nr.	Name	Default	Units	Definition
1	GAIN[1]	1.0×10^5		Open-loop gain
2	VOFF	0	V	Offset voltage
3	P1	1.0×10^2	Hz	Dominant pole frequency

Table 6-3. Op-amp (Linear 1-pole) Model Parameters

Nr.	Name	Default	Units	Definition
4	RIN	1.0×10^{7}	Ω	Input resistance
5	CMRR[1][2]	0		Common mode rejection ratio
6	M[3]	1		Device multiplier

1. Can also be specified in dB.
2. If *CMRR*=0, then 1/*CMRR* is ignored in the model equations.
3. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
yopa1 opamp1 n2 n1 n3 0 param: voff=100e-6
```

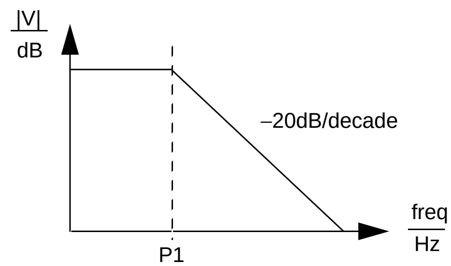
Specifies a linear gain operational amplifier yopa1 of model type opamp1 having input nodes n2 (+ve) and n1 (-ve) with output node n3. The op-amp offset voltage is declared explicitly in the instantiation line.

```
.model tdk modfas voff=100e-6
...
yopa1 opamp1d n1 n2 n3 n4 0 model: tdk
```

Specifies a differential linear gain operational amplifier yopa1 of type opamp1d with input nodes n1 (+ve) and n2 (-ve) and output nodes n3 (-ve) and n4 (+ve). The offset voltage parameter is declared using the **.MODEL** command.

Model Equations

Figure 6-5. Op-amp (Linear 1-pole) Model Characteristics



General

$$I(inp) = \frac{V(inp,inn) + VOFF}{RIN}$$

$$I(inn) = -\frac{V(inp,inn) + VOFF}{RIN}$$

$$TAU = \frac{1}{(2\pi \times \mathrm{P1})}$$

$$VA = GAIN \times \left((V(inp,inn) + VOFF) + \frac{1}{CMRR} \times \frac{(V(inp) + V(inn))}{2}\right)$$

Note

If *CMRR* = 0, then *1/CMRR* is ignored. *VA* is then calculated as if no value for *CMRR* had been specified.

Single Output

$$\frac{V(out,agnd)}{VA} = \frac{1}{1 + TAU \times p}$$

p is the complex frequency.

Differential Output

$$\frac{V(outn,agnd)}{VA} = -\frac{1}{2}\left(\frac{1}{1 + TAU \times p}\right)$$

$$\frac{V(outp,agnd)}{VA} = \frac{1}{2}\left(\frac{1}{1 + TAU \times p}\right)$$

Application Area

A typical application area for the linear one-pole op-amp is in the modeling of two-stage amplifiers including real saturation effects. This application is achieved in three parts as explained below:

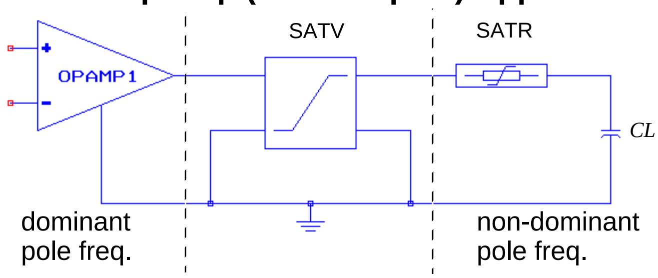
1. The first stage of the amplifier is described using the **`OPAMP1`** macromodel with the equation:

$$\frac{V_{out}}{V_{in}} = \frac{GAIN}{(1 + T \times p)}$$

2. Clipping of the voltage is achieved using the voltage limiter macromodel **`SATV`** at the output of the **`OPAMP1`** macromodel.

3. The second stage of the amplifier is described using an external capacitor `CL`, together with a non-linear resistor `R`, implemented as a saturating resistor (**`SATR`** macromodel).

Figure 6-6. Op-amp (Linear 1-pole) Application Area



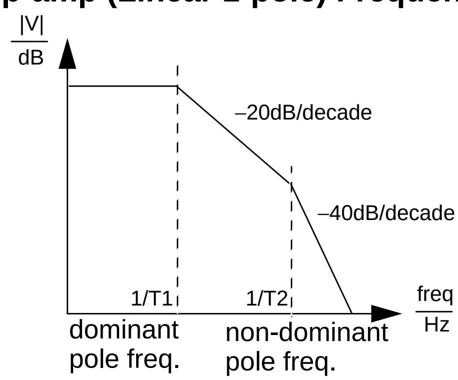
The dominant pole of the amplifier is determined by the following equation:

$$T1 = \frac{1}{(2\pi \times P1)} + R \times CL$$

The non-dominant pole of the amplifier is determined by the following equation:

$$T2 = \frac{1}{(2\pi \times P1)} \times (R \times CL)$$

Figure 6-7. Op-amp (Linear 1-pole) Frequency Response



An example of the above application can be found in "Examples" on page 1459.

Op-amp (Linear 2-pole)

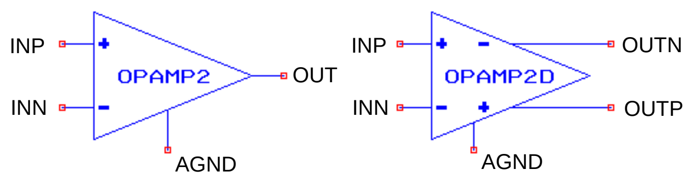
Single Output

```
Yxx OPAMP2 [PIN:] INP INN OUT AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Differential Output

```
Yxx OPAMP2D [PIN:] INP INN OUTN OUTP AGND
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-8. Op-amp (Linear 2-pole) Macromodel



Two linear 2-pole op-amp macromodels are provided, a single output linear gain two pole op-amp **OPAMP2**, and a differential output two pole op-amp **OPAMP2D**.

Model Pins

- INP

  Name of the positive input node

- INN

  Name of the negative input node

- OUT

  Output node for the single op-amp

- OUTP

  Positive output node for the differential output op-amp

- OUTN

  Negative output node for the differential output op-amp

- AGND

  Name of the ground node

Table 6-4. Op-amp (Linear 2-pole) Model Parameters

Nr.	Name	Default	Units	Definition
1	GAIN[1]	1.0×10^5		Open-loop gain
2	VOFF	0	V	Offset voltage
3	P1	1.0×10^2	Hz	Dominant pole frequency

Table 6-4. Op-amp (Linear 2-pole) Model Parameters

Nr.	Name	Default	Units	Definition
4	P2	1.0×10^6	Hz	Non-dominant pole frequency
5	RIN	1.0×10^7	Ω	Input resistance
6	CMRR[1,2]	0		Common mode rejection ratio
7	M[3]	1		Device multiplier

1. Can also be specified in dB.
2. If *CMRR*=0, then 1/*CMRR* is ignored in the model equations.
3. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
yopa1 opamp2 pin: n2 n1 n3 0 param: p1=300
```

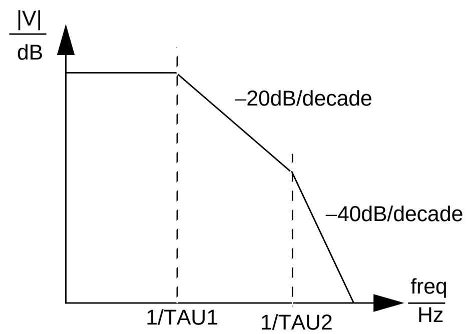
Specifies a linear two-pole operational amplifier yopa1 of model type opamp2 having input nodes n2 (+ve) and n1 (-ve) with output node n3. The op-amp dominant pole frequency parameter is declared explicitly in the instantiation line.

```
.model tdk modfas p1=300
...
yopa4 opamp2d n1 n2 n3 n4 0 model: tdk
```

Specifies a differential linear gain op-amp yopa4 of type opamp2d with input nodes n1 (+ve) and n2 (-ve) and output nodes n3 (-ve) and n4 (+ve). The dominant pole frequency parameter is declared using the **.MODEL** command.

Model Equations

Figure 6-9. Op-amp (Linear 2-pole) Model Characteristic



General

$$I(inp) = \frac{V(inp,inn) + VOFF}{RIN}$$

$$I(inn) = -\frac{V(inp,inn) + VOFF}{RIN}$$

$$\mathrm{TAU1} = \frac{1}{(2\pi \times \mathrm{P1})} + \frac{1}{(2\pi \times \mathrm{P2})}$$

$$\mathrm{TAU2} = \frac{1}{(2\pi \times \mathrm{P1})} \times \frac{1}{(2\pi \times \mathrm{P2})}$$

$$VA = GAIN \times \left((V(inp,inn) + VOFF) + \frac{1}{CMRR} \times \frac{(V(inp) + V(inn))}{2}\right)$$

Note

If *CMRR* = 0, then *1/CMRR* is ignored. *VA* is then calculated as if no value for *CMRR* had been specified.

Single output

$$\frac{V(out,agnd)}{VA} = \frac{1}{1 + \mathrm{TAU1} \times p + \mathrm{TAU2} \times p^2}$$

p is the complex frequency

Differential output

$$\frac{V(outn,agnd)}{VA} = -\frac{1}{2}\left(\frac{1}{1 + \mathrm{TAU1} \times p + \mathrm{TAU2} \times p^2}\right)$$

$$\frac{V(outp,agnd)}{VA} = \frac{1}{2}\left(\frac{1}{1 + \mathrm{TAU1} \times p + \mathrm{TAU2} \times p^2}\right)$$

Delay

```
DELxx IN OUT VAL
```

Figure 6-10. Delay Macromodel

This macromodel describes an ideal delay element that transfers its input voltage to its output after a specified time delay, where the reference node is ground. The input impedance is infinite.

Model Pins

- `IN`

  Name of the input node

- `OUT`

  Name of the output node

Parameters

- `VAL`

  Value of the delay in seconds. Must not be greater than `HMIN`.

Example

```
del1 a1 a2 2.0e-9
```

Specifies a delay element placed between nodes `a1` and `a2`, with a delay of 2ns.

Note

This macromodel can not be used in a **`.AC`** analysis.

Saturating Resistor

```
Yxx SATR [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-11. Saturating Resistor Macromodel

An analog saturating resistor macromodel. Current clipping can be specified directly using the parameter **IMAX**. This model is also designed to be used in conjunction with the one- and two-pole op-amp macromodels (**OPAMP1**, **OPAMP2**) and the voltage limiter **SATV**. If this option is used, the saturating current is specified indirectly by the value of the dominant pole frequency **P1** and the Slew Rate **SR**.

Model Pins

- IN

  Name of the input node

- OUT

  Name of the output node

Table 6-5. Saturating Resistor Model Parameters

Nr.	Name	Default	Units	Definition
1	R	1	Ω	Resistance
2	IMAX	1	A	Maximum current
3	SR	0	V/µs	Slew rate
4	P1	1.0×10^6	Hz	Dominant pole frequency
5	R1	30	Ω	Resistance of 1st order low pass filter
6	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

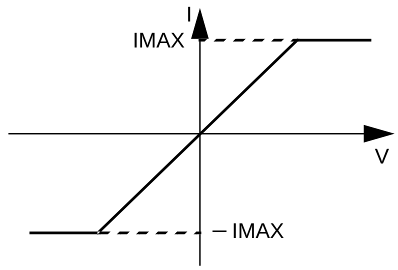
Example

```
ycl1 satr n1 n2
```

Specifies a current limiter ycl1 of type satr having input node n1 with output node n2. Default model parameters are used.

Model Equations

Figure 6-12. Saturating Resistor Model Characteristics



$$I = \frac{V(in) - V(out)}{R}$$

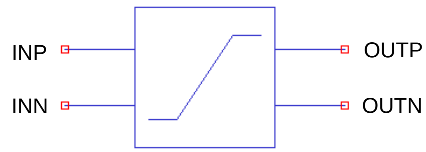
If ***SR*** is specified

$$IMAX = \frac{SR}{(2\pi \times \mathrm{P1} \times \mathrm{R1})}$$

Voltage Limiter

```
Yxx SATV [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-13. Voltage Limiter Macromodel



An analog voltage limiter macromodel, where the input voltage is limited to specified values. The voltage describes exponential behavior in the saturation regions and a 3rd degree polynomial in the working region.

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node

Table 6-6. Voltage Limiter Model Parameters

Nr.	Name	Default	Units	Definition
1	VMAX	5	V	Maximum output voltage
2	VMIN	–5	V	Minimum output voltage
3	VSATP	4.75	V	Positive saturation input voltage
4	VSATN	–4.75	V	Negative saturation input voltage
5	NSLOPE	0.25		Slope of Transfer Function at **VSATN**
6	PSLOPE	0.25		Slope of Transfer Function at **VSATP**
7	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
ysat1 satv n2 n1 n3 0 param: vmin=-3.0
```

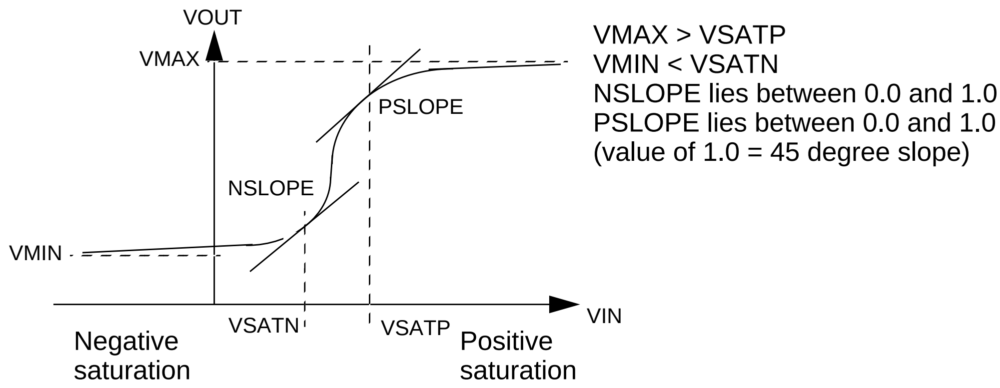
Specifies a voltage limiter ysat1 of type satv with input nodes n2 (+ve) and n1 (-ve) and output nodes n3 (+ve) and ground (–ve). The voltage below which negative saturation occurs is declared explicitly in the instantiation line.

```
.model satur modfas vmax=3.5
...
ys2 satv n1 n2 n3 0 model: satur
```

Specifies a voltage limiter ys2 of type satv having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and ground (-ve). The voltage above which positive saturation occurs is declared in the `.MODEL` command.

Model Equations

Figure 6-14. Voltage Limiter Model Characteristics



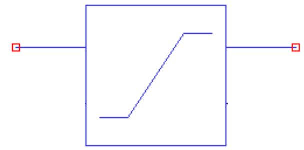
For $V(inp,inn) > VSATP \Rightarrow$ positive saturation.

For $V(inp,inn) < VSATN \Rightarrow$ negative saturation.

Current Limiter

```
Yxx SATI [PIN:] IN OUT
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-15. Current Limiter Macromodel



An analog current limiter macromodel, where the input current is limited to specified values. The current describes exponential behavior in the saturation regions and a 3rd degree polynomial in the working region.

Model Pins

- `IN`

  Name of the input node
- `OUT`

  Name of the output node

Table 6-7. Current Limiter Model Parameters

Nr.	Name	Default	Units	Definition
1	IMAX	5	A	Maximum output current
2	IMIN	–5	A	Minimum output current
3	ISATP	4.75	A	Positive saturation input current
4	ISATN	–4.75	A	Negative saturation input current
5	NSLOPE	0.25		Slope of Transfer Function at **ISATN**
6	PSLOPE	0.25		Slope of Transfer Function at **ISATP**
7	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
ysat1 sati n1 n2 param: imin=-3.0
```

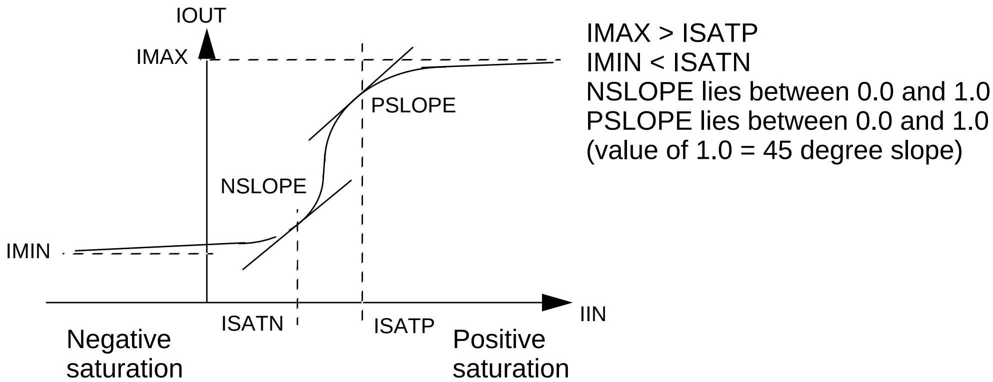
Specifies a current limiter ysat1 of type sati with input node n1 and output node n2. The current below which negative saturation occurs is declared explicitly in the instantiation line.

```
.model satur modfas imax=3.5
...
ys2 sati n1 n2 model: satur
```

Specifies a current limiter ys2 of type sati having input node n1 and output node n2. The current above which positive saturation occurs is declared in the **.MODEL** command.

Model Equations

Figure 6-16. Current Limiter Model Characteristics



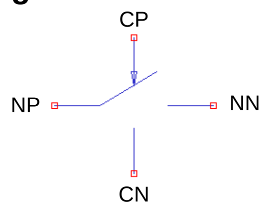
For $I(inp,inn) > ISATP \Rightarrow$ positive saturation.

For $I(inp,inn) < ISATN \Rightarrow$ negative saturation.

Voltage Controlled Switch

Yxx **VSWITCH** [**PIN:**] NP NN CP CN [**PARAM:** PAR=VAL {PAR=VAL}] **MODEL:** MNAME

Figure 6-17. Voltage Controlled Switch Macromodel



The voltage controlled switch macromodel can be thought of as a voltage controlled resistance, which varies continuously between the 'ON' and 'OFF' resistances defined by the model parameters **RON** and **ROFF**. The resistance change can be defined to be linear or exponential using the **LEVEL** parameter.

Recommendations for switch resistance values:

- Choose a reasonable ratio for **ROFF**/**RON**. A ratio smaller than 1×10^{12} (1/**GMIN**) is recommended.
- Make **RON** larger than **GMIN**. 1 Ω is a realistic value.
- Set **ROFF** as high as permissible with respect to the ratio and other circuit elements.

Model Pins

- NP

  Name of the input node
- NN

  Name of the output node
- CP

  Name of the positive controlling node
- CN

  Name of the negative controlling node

Table 6-8. Voltage Controlled Switch Model Parameters

Nr.	Name	Default	Units	Definition
1	LEVEL	1		Model level LEVEL=1— Linear interpolation LEVEL=2—Exponential interpolation LEVEL=3—Hysteresis behavior
2	VON	0.95	V	Control voltage for 'ON' state

Table 6-8. Voltage Controlled Switch Model Parameters

Nr.	Name	Default	Units	Definition
3	VOFF	0.05	V	Control voltage for 'OFF' state
4	RON	1.0×10^{-2}	Ω	'ON' resistance
5	ROFF	1.0×10^{10}	Ω	'OFF' resistance
6	HYSTERESIS	0	V	Switching Hysteresis
7	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Note

In AC mode, the switch resistance value is equal to that computed in the DC analysis.

Example

```
ysw1 vswitch n1 n4 n2 n3
```

Specifies a voltage controlled switch ysw1 of type vswitch having input and output nodes n1 and n4 respectively with a positive controlling node n2 and negative controlling node n3.

Model Equations

$$VCTRL = V(cp) - V(cn)$$

$$\mathrm{RC1} = ln\sqrt{RON \times ROFF}$$

$$\mathrm{RC2} = ln\left(\frac{RON}{ROFF}\right)$$

$$\mathrm{VC1} = \frac{(VON + VOFF)}{2}$$

$$\mathrm{VC2} = VOFF - VON$$

When $VON \geq VOFF$

$$VCTRL \geq VON \Rightarrow RS = RON$$

$VCTRL \leq VOFF \Rightarrow RS = ROFF$ where ***RS*** is the switch resistance.

$$VOFF < VCTRL < VON$$

For LEVEL=1

$$RS = \frac{ROFF - RON}{VOFF - VON} \times VCTRL + \frac{RON \times VOFF - (ROFF \times VON)}{VOFF - VON}$$

For LEVEL=2

$$RS = \exp\left(\mathrm{RC1} - \left(\frac{3 \times \mathrm{RC2} \times (VCTRL - \mathrm{VC1})}{2 \times \mathrm{VC2}}\right) + 2 \times \frac{\mathrm{RC2}(VCTRL - \mathrm{VC1})^3}{(\mathrm{VC2})^3}\right)$$

When $VON < VOFF$

$$VCTRL \le VON \Rightarrow RS = RON$$
$$VCTRL \ge VOFF \Rightarrow RS = ROFF$$
$$VOFF > VCTRL > VON$$

For LEVEL=3

$$VCTRL \le VON \Rightarrow RS = RON$$
$$VCTRL \ge VOFF \Rightarrow RS = ROFF$$

$RS =$

$$\frac{1}{\frac{1}{ROFF} + \frac{\left(\frac{1}{ROFF} - \frac{1}{RON}\right) \times (2(VC - VOFF)^3 - 3(\mathrm{VON\_HYS} - \mathrm{VOFF\_HYS}) \times (VC - \mathrm{VOFF\_HYS})^2)}{(\mathrm{VON\_HYS} - \mathrm{VOFF\_HYS})^3}}$$

for $VOFF > VCTRL > VON$

where:

$VOFF\_HYS = VOFF + HYSTERESIS \times HYSTERESIS\_STATE$

$VON\_HYS = VON + HYSTERESIS \times HYSTERESIS\_STATE$

$HYSTERESIS\_STATE = 1$, if VC waveform is rising and $VC < VOFF\_HYS$

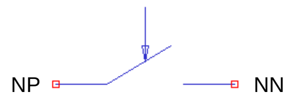
$HYSTERESIS\_STATE = -1$, if VC waveform is falling and $VC > VON\_HYS$

$HYSTERESIS\_STATE = 1$ initially

Current Controlled Switch

```
Yxx CSWITCH [PIN:] NP NN IC: VNAME
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-18. Current Controlled Switch Macromodel



The current controlled switch macromodel can be thought of as a current controlled resistance, where the resistance varies continuously between the 'ON' and 'OFF' resistances defined in the model parameters **RON** and **ROFF**. The resistance change can be defined to be linear or exponential using the **LEVEL** parameter.

Recommendations for switch resistance values:

- Choose a reasonable ratio for **ROFF**/**RON**. A ratio smaller than 1×10^{12} (1/**GMIN**) is recommended.
- Make **RON** larger than **GMIN**. 1 Ω is a realistic value.
- Set **ROFF** as high as permissible with respect to the ratio and other circuit elements.

Model Pins

- NP

  Name of the input node
- NN

  Name of the output node
- VNAME

  Controlling current through voltage source

Table 6-9. Current Controlled Switch Model Parameters

Nr.	Name	Default	Units	Definition
1	LEVEL	1		Model level **LEVEL**=1—Linear interpolation **LEVEL**=2—Exponential interpolation
2	ION	0.95	A	Control current for 'ON' state
3	IOFF	0.05	A	Control current for 'OFF' state
4	RON	1.0×10^{-2}	Ω	'ON' resistance
5	ROFF	1.0×10^{10}	Ω	'OFF' resistance

Table 6-9. Current Controlled Switch Model Parameters

Nr.	Name	Default	Units	Definition
6	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Note

In AC mode, the switch resistance value is equal to that computed in the DC analysis.

Example

```
ysw1 cswitch n1 n2 ic: v1
```

Specifies a current controlled switch ysw1 of type cswitch having input and output nodes n1 and n2 respectively. The switch is controlled by the current through the voltage source v1.

Model Equations

$$ICTRL = I(vname)$$

$$\mathrm{RC1} = ln\sqrt{RON \times ROFF}$$

$$\mathrm{RC2} = ln\left(\frac{RON}{ROFF}\right)$$

$$\mathrm{IC1} = \frac{(ION + IOFF)}{2}$$

$$\mathrm{IC2} = IOFF - ION$$

When $ION \geq IOFF$

$ICTRL \geq ION \Rightarrow RS = RON$
$ICTRL \leq IOFF \Rightarrow RS = ROFF$ where ***RS*** is the switch resistance.
$IOFF < ICTRL < ION$

For LEVEL=1

$$RS = \frac{ROFF - RON}{IOFF - ION} \times ICTRL + \frac{RON \times IOFF - (ROFF \times ION)}{IOFF - ION}$$

For LEVEL=2

$$RS = \exp\left(\mathrm{RC1} - \left(\frac{3 \times \mathrm{RC2} \times (ICTRL - \mathrm{IC1})}{2 \times IC2}\right) + 2 \times \frac{\mathrm{RC2}(ICTRL - \mathrm{IC1})^3}{(IC2)^3}\right)$$

When $ION < IOFF$

$$ICTRL \le ION \Rightarrow RS = RON$$
$$ICTRL \ge IOFF \Rightarrow RS = ROFF$$
$$IOFF > ICTRL > ION$$

For LEVEL=1

$$RS = \frac{ROFF - RON}{IOFF - ION} \times ICTRL + \frac{RON \times IOFF - (ROFF \times ION)}{IOFF - ION}$$

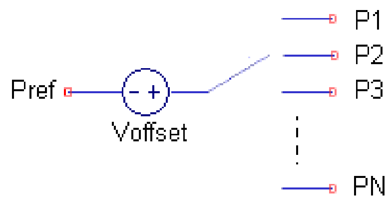
For LEVEL=2

$$RS = \exp\left(\mathrm{RC1} - \left(\frac{3 \times \mathrm{RC2} \times (ICTRL - IC1)}{2 \times IC2}\right) + 2 \times \frac{\mathrm{RC2}(ICTRL - IC1)^3}{(IC2)^3}\right)$$

Ideal Single-Pole Multiple-Throw Switch

```
Yxx IDEAL_SW [PIN:] Pref [P1 {Pn}] [PARAM: PAR=VAL {PAR=VAL}]
+ [MODEL: MNAME]
```

Figure 6-19. Ideal Single-Pole Multiple-Throw Switch Macromodel



The ideal single-pole multiple-throw switch macromodel is an ideal switch with zero on, infinite off resistance. The switch can change its position based on which simulation analysis type is being performed and the switch state is allowed to change only between different simulation analyses.

Model Pins

- `Pref`

  Name of the reference node
- `P1`

  Name of the first switch node
- `Pn`

  Name of the n^{th} switch node

Table 6-10. Ideal Single-Pole Multiple-Throw Switch Parameters

Nr.	Name	Default	Units	Definition
1	POSITION	0		Switch position (0, 1, 2, ...)
2	DC_POSITION	0		Position to which switch is set at start of DC analysis
3	AC_POSITION	0		Position to which switch is set at start of AC analysis
3	DCAC_POSITION	0		Position to which switch is set at start of DCAC analysis (the DC analysis which runs before, and as part of, AC analysis)
4	TRAN_POSITION	0		Position to which switch is set at start of transient analysis

Table 6-10. Ideal Single-Pole Multiple-Throw Switch Parameters

Nr.	Name	Default	Units	Definition
5	IC_POSITION	0		Position to which switch is set at start of IC analysis (the DC analysis which runs before, and as part of, TRANSIENT analysis)
6	OFFSET	0	V	Offset voltage in series with common terminal

- if POSITION=0, no terminal is connected to any other
- if POSITION=n, terminal Pn is connected to Pref
- if POSITION=n and OFFSET=VAL1, terminal Pn is connected to Pref through a voltage source of value=VAL1
- if XX_POSITION=n, terminal Pn is connected to Pref in XX analysis only
- Analysis specific position parameter XX_POSITION will always dominate over a position given with the POSITION parameter during the XX analysis
- The XX_POSITION parameters should be used carefully; careless use can generate discontinuities that result in convergence problems

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Example

```
ysw1 vswitch n1 n4 n2 n3
```

Specifies a voltage controlled switch ysw1 of type vswitch having input and output nodes n1 and n4 respectively with a positive controlling node n2 and negative controlling node n3.

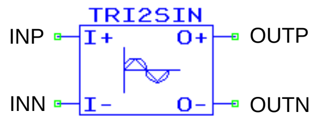
```
ysw1 ideal_sw Pref P1 P2 PARAM: POSITION=2 OFFSET=1
```

Specifies an ideal switch ysw1 of type ideal_sw having reference node Pref, and two other switch terminals P1, P2. Since position is set to 2 with Offset voltage 1Volt, then the reference node will be connected to node P2 through a voltage source of value=1Volt, while port P1 will be left un-connected.

Triangular to Sine Wave Converter

```
Yxx TRI2SIN [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-20. Triangular to Sine Wave Converter Macromodel



This analog macromodel converts a triangular wave into a sinusoidal wave.

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node

Table 6-11. Triangular to Sine Wave Converter Model Parameters

Nr.	Name	Default	Units	Definition
1	GAIN	1		Gain
2	VOFF	0	V	Offset voltage
3	VU	1	V	Upper limit of input voltage
4	VL	–1	V	Lower limit of input voltage
5	LEVEL	1		Model index
6	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
ytr1 tri2sin pin: n1 0 n2 0 param: vu=0.0 vl=8
```

Specifies a triangular to sine wave converter ytr1 of type tri2sin having input nodes n1 (+ve) and ground (-ve) with output nodes n2 (+ve) and ground (-ve). The upper and lower limits of the input voltage are declared explicitly in the instantiation line.

```
.model sto modfas voff=0.2
...
ytr2 tri2sin n1 n2 n3 n4 param: vu=5 model: sto
```

Specifies a triangular to sine wave converter ytr2 of type tri2sin having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The offset voltage is declared using the **.MODEL** command.

Model Characteristics

A mathematical transformation is used to convert the triangular input to the sine output. Input signal frequency, maximum and minimum voltage are all taken into account.

If **LEVEL**=1 is specified (default value), the model assumes that the maximum input voltage is equal to **VU** and that the minimum input voltage is equal to **VL**.

If **LEVEL**=2 is specified, the model itself calculates **VU** and **VL** at runtime.

$$VU = max(V(inp,inn))$$

$$VL = min(V(inp,inn))$$

$$V(outp,outn) = VOFF + VS + GAIN \times VM \times \sin(K)$$

where

$$K = (V(inp,inn) - VS) \times \frac{\pi}{VM/2}$$

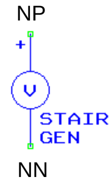
$$VM = \frac{(VU - VL)}{2}$$

$$VS = \frac{(VU + VL)}{2}$$

Staircase Waveform Generator

```
Yxx STAIRGEN [PIN:] NP NN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-21. Staircase Waveform Generator Macromodel



A staircase waveform generator macromodel.

Model Pins

- `NP`

  Name of the positive input nodes

- `NN`

  Name of the negative input nodes

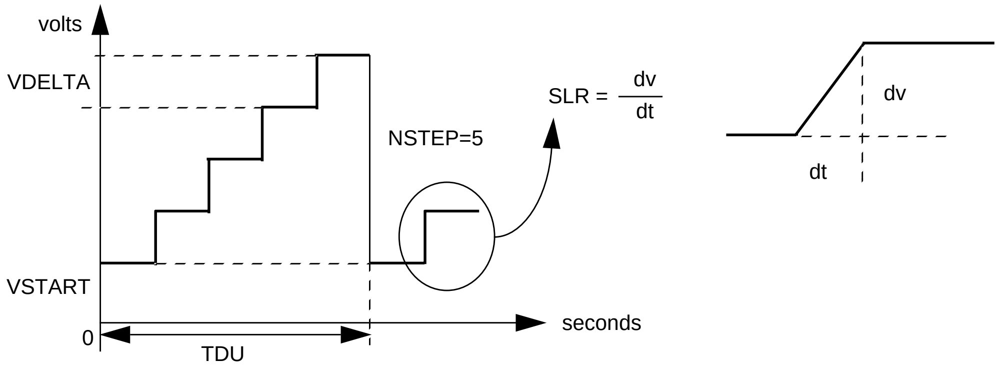
Table 6-12. Staircase Waveform Generator Model Parameters

Nr.	Name	Default	Units	Definition
1	VSTART	0	V	Start voltage
2	VDELTA	0.1	V	Step voltage
3	NSTEP	10		Number of voltage steps
4	TDU	1.0×10^{-4}	s	Period duration time
5	SLR	1	V/µs	Slew rate for falling and rising edges
6	M[1]	1		Device multiplier

1. **`.OPTION YMFACT`** must be specified for **`M`** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Figure 6-22. Staircase Waveform Generator Model Characteristics



Note

The parameter set has to be consistent with the following restriction on Slew Rate:

$$SLR \geq 200 \times (NSTEP - 1) \times \frac{VDELTA}{TDU}$$

Examples

```
ytr1 stairgen pin: n1 n2 param: vstart=0.0 nstep=8
```

Specifies a staircase voltage generator ytr1 of type stairgen having input node n1 with output node n2. The start and step voltage are declared explicitly in the instantiation line.

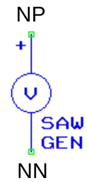
```
.model sto modfas nstep=5.0
...
ytr2 stairgen n1 n2 param: vstart=1.0 vdelta=0.2 model: sto
```

Specifies a staircase voltage generator ytr2 of type stairgen having input nodes n1 (+ve) and n2 (-ve). The number of voltage steps is declared using the `.MODEL` command.

Sawtooth Waveform Generator

```
Yxx SAWGEN [PIN:] NP NN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-23. Sawtooth Waveform Generator Macromodel



A sawtooth waveform generator macromodel.

Model Pins

- `NP`

  Name of the positive input node

- `NN`

  Name of the negative input node

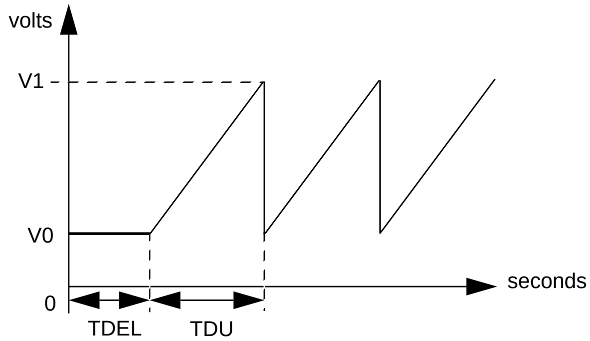
Table 6-13. Sawtooth Waveform Generator Model Parameters

Nr.	Name	Default	Units	Definition
1	V0	0	V	Initial voltage
2	V1	5	V	Voltage magnitude
3	TDU	1.0×10^{-4}	s	Duration of sawtooth
4	TDEL	0	s	Delay time
5	M[1]	1		Device multiplier

1. **`.OPTION YMFACT`** must be specified for **`M`** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Figure 6-24. Sawtooth Waveform Generator Model Characteristics



Examples

```
ytr1 sawgen pin: n1 n2 param: tdel=0.001
```

Specifies a sawtooth waveform generator ytr1 of type sawgen having input node n1 with output node n2. The delay time is declared explicitly in the instantiation line.

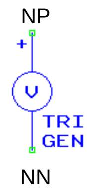
```
.model sto modfas tdel=0.0001 v0=1.0
...
ytr2 sawgen n1 n2 param: tdel=1.0 model: sto
```

Specifies a sawtooth waveform generator ytr2 of type sawgen having input node n1 and output node n2. The delay time and initial voltage are declared using the **.MODEL** command. Note that the delay time declared in the instantiation line overrides the delay time parameter in the **.MODEL** command.

Triangular Waveform Generator

`Yxx TRIGEN [PIN:] NP NN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]`

Figure 6-25. Triangular Waveform Generator Macromodel



A triangle waveform generator macromodel.

Model Pins

- `NP`

  Name of the positive input node

- `NN`

  Name of the negative input node

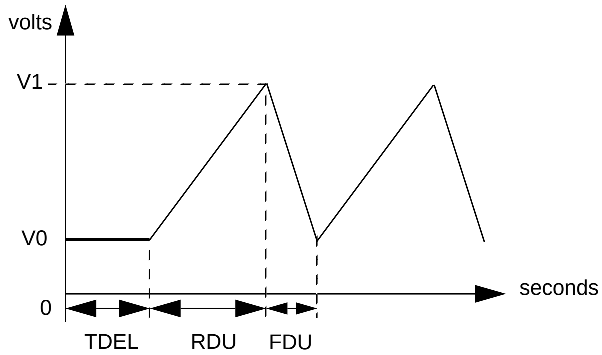
Table 6-14. Triangular Waveform Generator Model Parameters

Nr.	Name	Default	Units	Definition
1	V0	0	V	Initial voltage
2	V1	5	V	Voltage magnitude
3	RDU	1.0×10^{-4}	s	Duration of first edge
4	FDU	1.0×10^{-4}	s	Duration of second edge
5	TDEL	0	s	Delay time
6	M[1]	1		Device multiplier

1. **`.OPTION YMFACT`** must be specified for **`M`** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Figure 6-26. Triangular Waveform Generator Model Characteristics



Examples

```
ytr1 trigen pin: n1 n2 param: tdel=0.001
```

Specifies a sawtooth waveform generator ytr1 of type trigen having input node n1 with output node n2. The delay time is declared explicitly in the instantiation line.

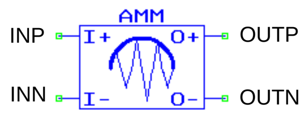
```
.model sto modfas tdel=0.0001 v0=1.0
...
ytr2 trigen n1 n2 param: rdu=1.0e-3 model: sto
```

Specifies a sawtooth waveform generator ytr2 of type trigen having input node n1 and output node n2. The delay time and initial voltage are declared using the **.MODEL** command.

Amplitude Modulator

`Yxx AMM [PIN:] INP INN OUTP OUTN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]`

Figure 6-27. Amplitude Modulator Macromodel



An amplitude modulator macromodel. The carrier frequency is specified with the **FC** parameter and the type of carrier signal is specified by **LEVEL** parameter (**LEVEL**=1— sinusoidal waveform, **LEVEL**=2— pulse waveform). The modulation input at lower frequencies is applied at the inp and inn terminals. The **VOFF** parameter controls the modulation depth and the final peak-to-peak voltage.

Model Pins

- INP

  Name of the positive input node
- INN

  Name of the negative input node
- OUTP

  Name of the positive output node
- OUTN

  Name of the negative output node

Table 6-15. Amplitude Modulator Model Parameters

Nr.	Name	Default	Units	Definition
1	LEVEL	1		Type of carrier signal **1**—sinusoidal, **2**—pulse
2	SLR	10	V/µs	Slew rate
3	VOFF	0	V	Offset voltage
4	FC	1.0×10^6	Hz	Carrier frequency
5	NSAM	10		Each period of the carrier signal is sampled by at least **NSAM** points
6	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
ya1 amm pin: n1 n2 n3 n4 param: voff=0.5
```

Specifies an amplitude modulator ya1 of type amm having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The offset voltage is declared explicitly in the instantiation line.

```
.model sto modfas fc=1u
...
ya2 amm n1 n2 n3 n4 model: sto
```

Specifies an amplitude modulator ya2 of type amm having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The carrier frequency is declared using the **.MODEL** command.

Model Characteristics

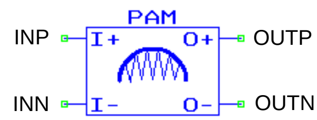
- LEVEL=1

$$V(outp, outn) = (V(inp, inn) + VOFF) \times \sin(2\pi \times TIME \times FC)$$

Pulse Amplitude Modulator

```
Yxx PAM [PIN:] INP INN OUTP OUTN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-28. Pulse Amplitude Modulator Macromodel



A pulse amplitude modulator macromodel. The carrier frequency is specified with the **FC** parameter and the type of carrier signal is specified by the **LEVEL** parameter (**LEVEL**=1—sinusoidal waveform, **LEVEL**=2—pulse waveform). The modulation input at lower frequencies is applied at the `inp` and `inn` terminals. The modulation input can be centered about zero, or can be set exclusively positive or negative. The **VOFF** parameter controls the modulation depth by offsetting the modulation input.

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node

Table 6-16. Pulse Amplitude Modulator Model Parameters

Nr.	Name	Default	Units	Definition
1	LEVEL	1		Type of carrier signal 1—sinusoidal, 2—pulse
2	SLR	10	V/µs	Slew rate
3	VOFF	0	V	Offset voltage
4	FC	1.0×10^6	Hz	Carrier frequency
5	NSAM	10		Each period of the carrier signal is sampled by at least **NSAM** points
6	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
yp1 pam pin: n1 n2 n3 n4 param: voff=0.5
```

Specifies a pulse amplitude modulator yp1 of type pam having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The offset voltage is declared explicitly in the instantiation line.

```
.model sto modfas fc=1u
...
yp2 pam n1 n2 n3 n4 model: sto
```

Specifies a pulse amplitude modulator yp2 of type pam having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The carrier frequency is declared using the **.MODEL** command.

Model Characteristics

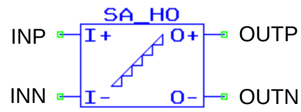
- LEVEL=1

$$V(outp,outn) = \frac{V(inp,inn) + VOFF}{2} \times (1 + \sin(2\pi \times TIME \times FC))$$

Sample and Hold

```
Yxx SA_HO [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-29. Sample & Hold Macromodel



A Sample and Hold circuit macromodel. At each sampling point the output voltage is fixed to the level of the input voltage. After the acquisition time **TACQ** has elapsed, the output voltage is kept at the level of the input voltage for a period of 1/**FS** until the next sampling point.

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node

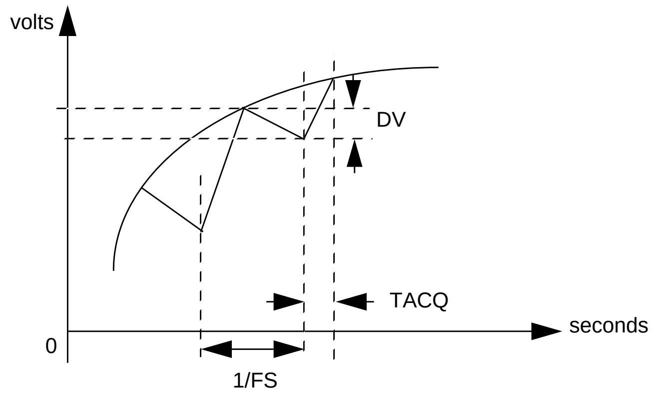
Table 6-17. Sample & Hold Model Parameters

Nr.	Name	Default	Units	Definition
1	FS	1.0×10^{6}	Hz	Sampling frequency
2	TACQ	1.0×10^{-9}	s	Acquisition time
3	DV	0.02	V	Droop voltage
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Figure 6-30. Sample & Hold Model Characteristics



Examples

```
ys1 sa_ho pin: n1 n2 n3 n4 param: fs=5meg
```

Specifies a Sample and Hold macromodel ys1 of type sa_ho with input nodes n1 (+ve) and n2 (-ve) and output nodes n3 (+ve) and n4 (-ve). The sampling frequency is declared explicitly in the instantiation line.

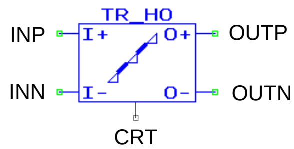
```
.model sto modfas dv=0.05
...
ys2 sa_ho n1 n2 n3 n4 model: sto
```

Specifies a Sample and Hold ys2 of type sa_ho having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The droop voltage is declared using the **.MODEL** command.

Track and Hold

```
Yxx TR_HO [PIN:] INP INN OUTP OUTN CRT
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-31. Track & Hold Macromodel



A Track and Hold circuit macromodel. The model has two modes of operation depending on the logic level of the `CRT` voltage. Upon receiving the `CRT` pulse, the model swings the output voltage towards the input voltage and forces the output voltage to follow (or track) the input voltage for the remainder of the pulse. This is called the *Track Mode*. After the S/H pulse is removed, the model holds the output voltage at the value that the input voltage had at the instant of pulse deactivation. This is called the *Hold Mode*.

Model Pins

- `INP`

  Name of the positive input node

- `INN`

  Name of the negative input node

- `OUTP`

  Name of the positive output node

- `OUTN`

  Name of the negative output node

- `CRT`

  Name of the controlling voltage node

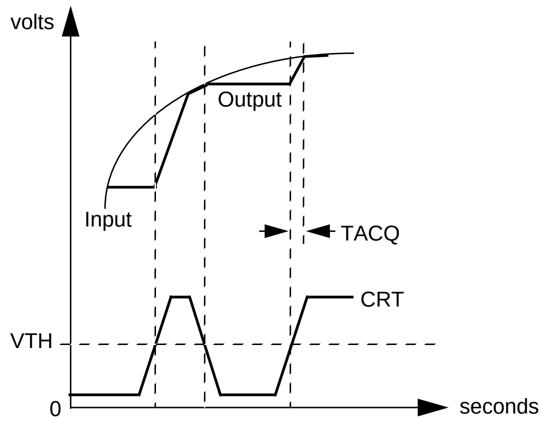
Table 6-18. Track & Hold Model Parameters

Nr.	Name	Default	Units	Definition
1	VTH	0.5	V	Threshold voltage for node `CRT`
2	TACQ	1.0×10^{-9}	s	Acquisition time
3	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Figure 6-32. Track & Hold Model Characteristics



Example

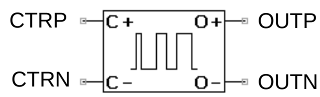
```
yt1 tr_ho pin: n1 n2 n3 n4 param: tacq=1u
```

Specifies a Track and Hold macromodel yt1 of type tr_ho having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve). The acquisition time is declared explicitly in the instantiation line.

Pulse Width Modulator

```
Yxx PWM [PIN:] CTRP CTRN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-33. Pulse Width Modulator Macromodel



This macromodel generates a pulse width modulated signal. The duty cycle of the pulse width modulator signal can be controlled in a specified range by the input voltage.

Model Pins

- `CTRP`

  Name of the positive control node
- `CTRN`

  Name of the negative control node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node

Table 6-19. Pulse Width Modulator Model Parameters

Nr.	Name	Default	Units	Definition
1	DUTYMIN	0.001		Minimum duty cycle
2	DUTYMAX	0.999		Maximum duty cycle
3	CTRLMIN	0	V	Minimum control voltage
4	CTRLMAX	1	V	Maximum control voltage
5	PFREQ	1.0×10^{3}	Hz	Pulse frequency
6	TR	1.0×10^{-6}	s	Pulse rise time
7	TF	1.0×10^{-6}	s	Pulse fall time
8	NDELAY	0		Number of delay pulse cycles
9	PVMIN	0	V	Low pulse voltage
10	PVMAX	1	V	High pulse voltage
11	M[1]	1		Device multiplier

1. `.OPTION YMFACT` must be specified for `M` to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

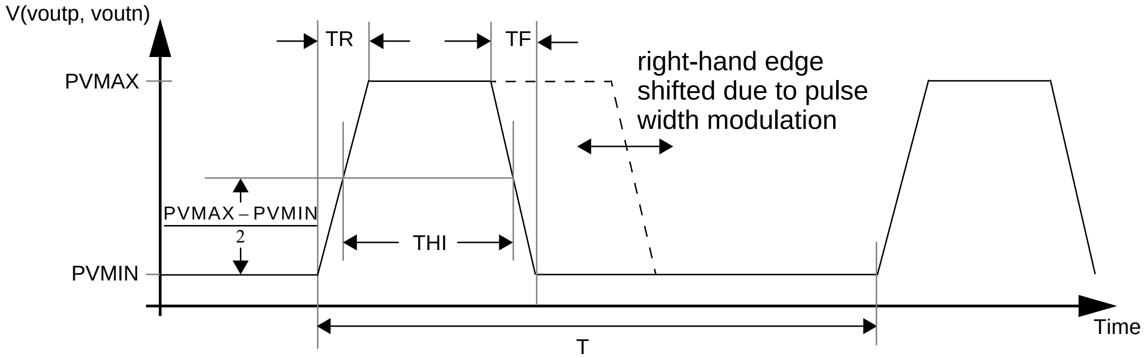
Example

```
yp1 pwm pin: n1 0 n3 0 param: pvmax=5.0
```

Specifies a pulse width modulator macromodel yp1 of type pwm having input nodes n1 (+ve control) and ground (-ve control) with output nodes n3 (+ve output) and ground (-ve output). The high pulse voltage is declared explicitly in the instantiation line.

Model Characteristics

Figure 6-34. Pulse Width Modulator Model Characteristics



Note

For correct operation of the macromodel, the following constraints must be considered: The duty cycle of the output pulse signal can only be in the following range:

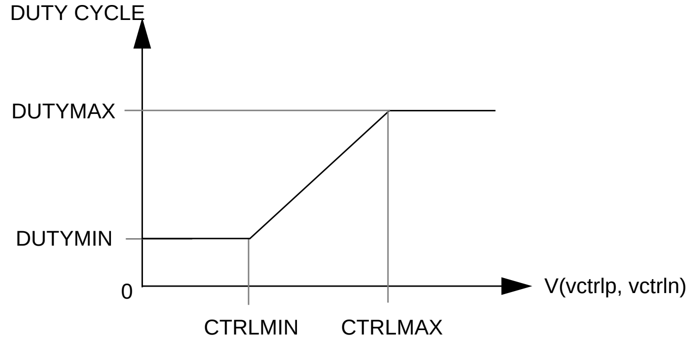
$$DUTYCYCLE = \frac{((TR)/2 + (TF)/2)}{T} \ldots \frac{(T - (TR)/2 - (TF)/2)}{T}$$

with:

$$T = \frac{1}{PFREQ}$$

$$DUTYCYCLE = \frac{THI}{T}$$

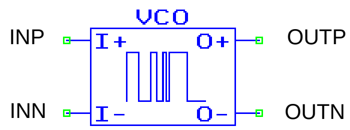
Figure 6-35. Pulse Width Modulator Duty Cycle



Voltage Controlled Oscillator

```
Yxx VCO [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-36. Voltage Controlled Oscillator Macromodel



On the output nodes a signal is generated whose frequency is dependent on the input voltage. The **LEVEL** parameter specifies the type for the waveform, either sinusoidal or pulse.

Model Pins

- `INP`

  Name of the positive input node

- `INN`

  Name of the negative input node

- `OUTP`

  Name of the positive output node

- `OUTN`

  Name of the negative output node

Table 6-20. Voltage Controlled Oscillator Model Parameters

Nr.	Name	Default	Units	Definition
1	V1	1	V	Output amplitude
2	VOFF	0	V	Offset voltage
3	FMIN	1	Hz	Minimum allowed frequency—only for LEVEL=2, 3
4	FMAX	1.0×10^{10}	Hz	Maximum allowed frequency—only for LEVEL=2, 3
5	LEVEL	1		**LEVEL**=1—Continuous sinusoidal **LEVEL**=2—Sampled sinusoidal **LEVEL**=3—Sampled pulse
6	A	0	Hz	Polynomial parameter
7	B	1	Hz/V	Polynomial parameter
8	C	0	Hz/V^2	Polynomial parameter
9	D	0	Hz/V^3	Polynomial parameter
10	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
y1 vco pin: n1 n2 n3 n4 param: voff=0.5
```

Specifies a vco model y1 having input nodes n1 (+ve) and n2 (-ve), with output nodes n3 (+ve) and n4 (-ve). The offset voltage parameter is declared explicitly in the instantiation line.

```
.model tdk modfas FMIN=10k
...
ys4 vco n1 n2 n3 0 model: tdk
```

Specifies a vco model having input nodes n1 (+ve) and n2 (-ve) with output node n3 (+ve). The **FMIN** parameter is declared using the **.MODEL** command.

Model Equations

$$vin = V(inp,inn)$$

VCO Frequency

$$f = a + b \times vin + c \times vin^2 + d \times vin^3$$

$$V(outp, outn) = voff + v1 \times \sin(2\pi \times TIME \times f)$$

- Level=1

  The frequency of the sinusoidal output signal is determined at each internal time step.

- Level=2

  The frequency of the sinusoidal output is determined only once for one period of the output signal.

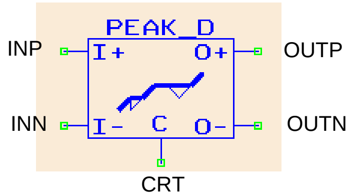
- Level=3

  The frequency of the pulse output is determined only once for one period of the output signal.

Peak Detector

```
Yxx PEAK_D [PIN:] INP INN OUTP OUTN CRT
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-37. Peak Detector Macromodel



Positive or negative peak detection is selected using the **LEVEL** parameter. The peak detector tracks the input signal and hold the output at the highest (or lowest) peak found since operation of the CRT voltage or the **RES** parameter. The input waveform is continuously compared with the stored peak value to determine whether the stored value should be updated.

Model Pins

- INP

  Name of the positive input node
- INN

  Name of the negative input node
- OUTP

  Name of the positive output node
- OUTN

  Name of the negative output node
- CRT

  Name of the control node

Table 6-21. Peak Detector Model Parameters

Nr.	Name	Default	Units	Definition
1	VTH	1	V	Threshold voltage—signal on CRT pin
2	RES	0.5	V	If the output voltage crosses the RES value the output is reset to 0
3	SLR	1	V/µs	The output signal follows the input signal with the slewrate **SLR**
4	RSLR	1	V/µs	Reset to 0 with the slewrate **RSLR**
5	LEVEL	1		**LEVEL**=1—positive peak detector **LEVEL**=2—negative peak detector

Table 6-21. Peak Detector Model Parameters

Nr.	Name	Default	Units	Definition
6	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
y1 peak_d pin: n1 n2 n3 n4 n5 param: level=2 res=-5.0
```

Specifies a `peak_d` model y1 having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve) and n4 (-ve) and control node n5. The **level** and **res** parameters are declared explicitly in the instantiation line.

```
.model tdk modfas SLR=10
...
ys4 peak_d n1 n2 n3 0 0 model:tdk
```

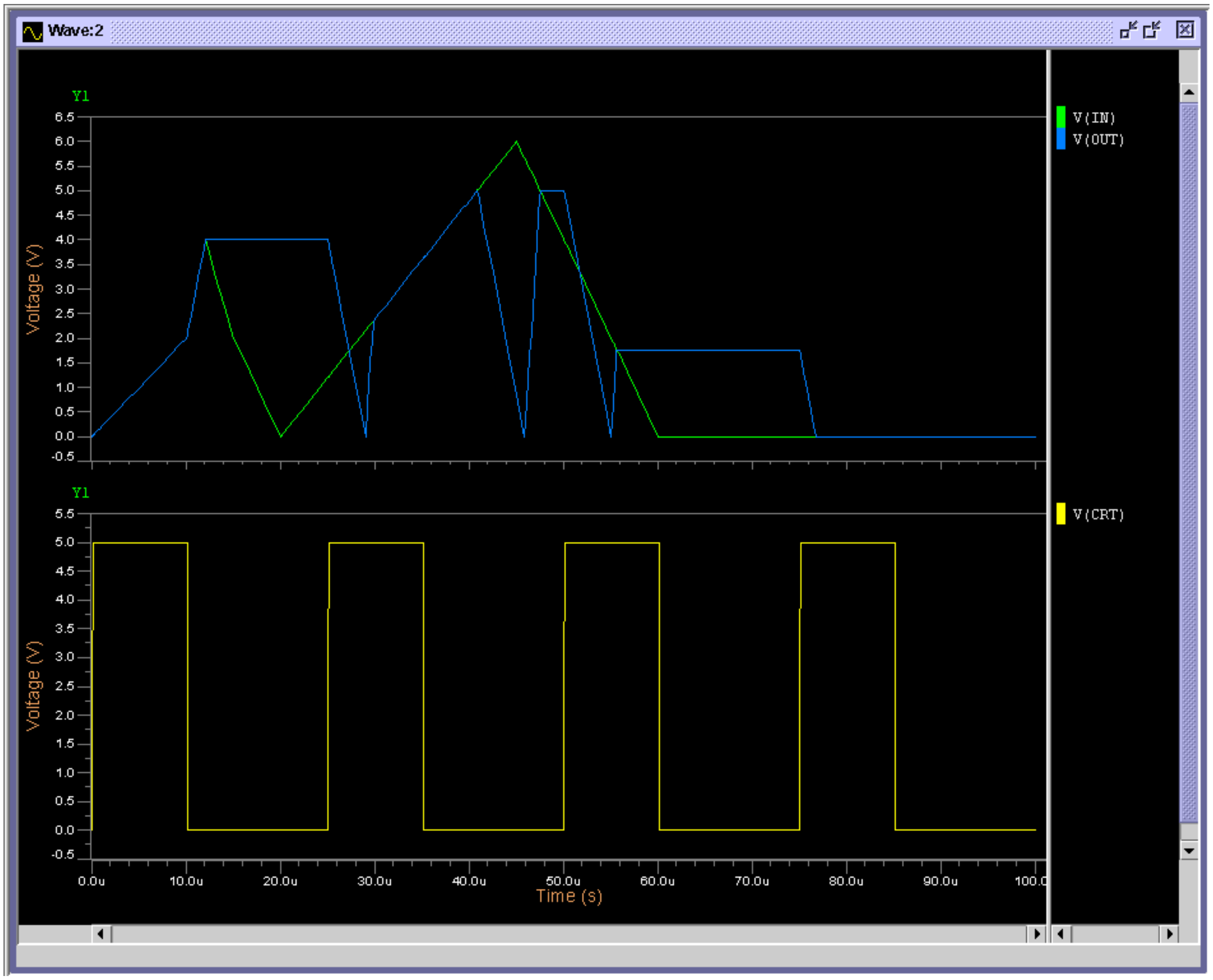
Specifies a `peak_d` model having input nodes `n1` (+ve) and `n2` (–ve) with output node `n3` (+ve). The **SLR** parameter is declared using the **.MODEL** command.

```
.MODEL TDK_PEAK_D MODFAS VTH=1 res=4
Vin in 0 pwl 0 0 10u 2 12u 4 15u 2 20u 0 45u 6 60u 0
Vcrt crt 0 pulse 0 5 0 .1u .1u 10u 25u
*Vcrt crt 0 5
Y26 PEAK_D PIN: IN 0 OUT 0 CRT PARAM: RES=5 SLR=3.0 RSLR=1.0 LEVEL=1.0
MODEL: TDK_PEAK_D
rout out 0 1k
.tran 1u 100u
.plot tran v(in)
.plot tran v(crt)
.plot tran v(out)
.end
```

In this example—see also simulation results over page—the peak detector tracks the input signal V(IN) and holds the output V(OUT) at the highest value of 4V (seen at 12µs). The rising voltage on the control node V(CRT) at 25µs causes the V(OUT) voltage to fall by 1V/µs. The output then tracks the input (from 29µs) until it reaches the RES value (5V) at 40.8µs and falls back to 0.

At 46 µs V(OUT) follows V(IN) until 50µs has passed, whereupon the input at V(CRT) causes the V(OUT) signal to return to 0. Finally, at 55µs, V(OUT) stays at the lowest V(IN) peak until an impulse on V(CRT) at 75µs causes the signal to fall to 0.

Figure 6-38. Peak Detector Simulation Results



Level Detector

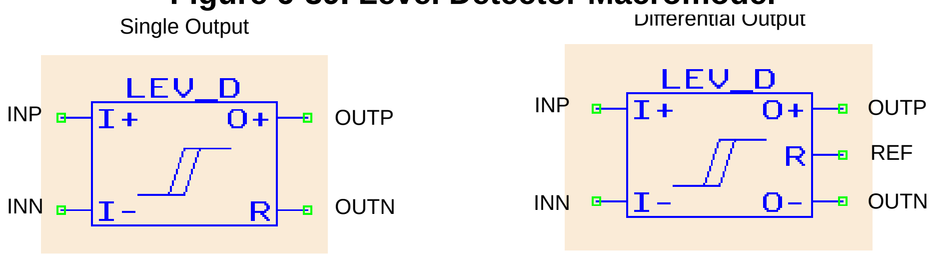
Single Output

```
Yxx LEV_D [PIN:] INP INN OUTP OUTN
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Differential Output

```
Yxx LEV_D [PIN:] INP INN OUTP OUTN REF
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-39. Level Detector Macromodel



The model is used to convert analog signals into bi-level signals. This is done by comparing an input signals with a reference value. Depending on the way that the parameters and number of nodes are chosen, the model works as an inverting or non-inverting zero-crossing or level detector with single or differential output and with or without hysteresis.

Model Pins

- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node
- `REF`

  Name of the reference node. Only used for differential output.

Table 6-22. Level Detector Model Parameters

Nr.	Name	Default	Units	Definition
1	TR	1	µs	Time for the output signal switching from **V0** to **V1**
2	TF	1	µs	Time for the output signal switching form **V1** to **V0**

Table 6-22. Level Detector Model Parameters

Nr.	Name	Default	Units	Definition
3	TPD	0	s	Transit time through the comparator
4	V0	0	V	Lower voltage level
5	V1	1	V	Higher voltage level
6	VOFF	0	V	Is added to the potential at the node `INN`
7	VRL	–0.1	V	Lower reference voltage
8	VRU	0.1	V	Higher reference voltage
9	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
y1 lev_d pin: n1 n2 n3 0 param: v1=5 vrl=2.4 vru=2.6
```

Specifies a lev_d model y1 with single output having input nodes n1 (+ve) and n2 (-ve) with output nodes n3 (+ve). Parameters v1, vrl and vru are declared explicitly in the instantiation line.

```
.model tdk modfas vrl=0.0 vru=0.0
...
ys4 lev_d n1 n2 n3 n4 n5 model:tdk
```

Specifies a lev_d model with differential output having input nodes n1 (+ve) and n2 (-ve), with non-inverting output node n3 and inverting output node n4. As reference for the output node n5 is used. The vrl and vru parameters are declared via the **.MODEL** command. The above model works as a differential zero-crossing detector without hysteresis.

Model Equations

If a rising input voltage $VIN = V(INP) - V(INN) - VOFF$ passes the lower reference voltage, VRL, the output signal switches from V0 to V1 during the time TR.

If a falling input voltage $VIN = V(INP) - V(INN) - VOFF$ passes the upper reference voltage, VRU, the output signal switches from V1 to V0 during the time TF.

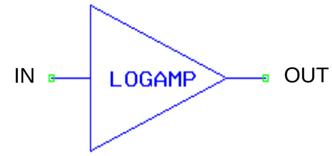
The 4 node model works as a non-inverting comparator, the output signal being applied across `OUTP` and `OUTN`. The 5 node model works as a differential comparator. The non-inverting output signal is applied across `OUTP` and `REF`, and the inverting output signal is applied across `OUTN` and `REF`.

If the parameters VRL and VRU have the same value the model operates as zero-crossing detector.

Logarithmic Amplifier

```
Yxx LOGAMP [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-40. Logarithmic Amplifier Macromodel



The model provides a logarithmic transfer function between the input and the output node. The signal on the output node is limited by user defined values.

Model Pins

- `IN`

  Name of the input node
- `OUT`

  Name of the output node

Table 6-23. Logarithmic Amplifier Model Parameters

Nr.	Name	Default	Units	Definition
1	K	1		Gain
2	E	1	V	Influences the argument of the log function
3	BASE	e—natural logarithm		Base of the logarithm
4	VMAX	5	V	Maximum output voltage
5	VMIN	–5	V	Minimum output voltage
6	VSATP	4.75	V	Positive saturation voltage
7	VSATN	–4.75	V	Negative saturation voltage
8	PSLOPE	0.25		Slope of Transfer Function at VSATP
9	NSLOPE	0.25		Slope of Transfer Function at VSATN
10	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Examples

```
y1 logamp pin: n1 n2 param: K=100.0
```

Specifies a logamp model y1 having input node n1 with output node n2. The gain parameter is declared explicitly in the instantiation line.

```
.model tdk modfas VMAX=10 VMIN=-10 VSATP=9.9 VSATN=-9.9
...
ys4 logamp n1 n2 model:tdk
```

Specifies a logamp model having input node n1 with output node n2. Parameters VMAX, VMIN, VSATP and VSATN are declared using the **.MODEL** command.

Model Equations

$$vi = v(IN)$$

if:

$$vi \le 1.0\times 10^{-10}$$

then:

$$vi = 1.0\times 10^{-10}$$

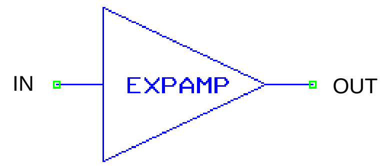
$$v(OUT) = LIMITER\left\{-K \times \frac{1}{(\log BASE)} \times \log\left(\frac{vi}{E}\right)\right\}$$

Information on voltage limiting is given in the "Voltage Limiter" on page 397. Non-limited voltages can be specified using `v(yname->lin)`.

Anti-logarithmic Amplifier

```
Yxx EXPAMP [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-41. Anti-logarithmic Amplifier Macromodel



The model provides an anti-logarithmic transfer function between the input and the output node. The signal on the output node is limited by user defined values.

Model Pins

- `IN`

  Name of the input node
- `OUT`

  Name of the output node

Table 6-24. Anti-logarithmic Amplifier Model Parameters

Nr.	Name	Default	Units	Definition
1	K	1		Gain
2	E	1	V	Influences the argument of the power function
3	BASE	e exponential		Base of the power function
4	VMAX	5	V	Maximum output voltage
5	VMIN	–5	V	Minimum output voltage
6	VSATP	4.75	V	Positive saturation voltage
7	VSATN	–4.75	V	Negative saturation voltage
8	PSLOPE	0.25		Slope of Transfer Function at VSATP
9	NSLOPE	0.25		Slope of Transfer Function at VSATN
10	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
y1 expamp pin: n1 n2 param: K=100.0
```

Specifies an expamp model y1 having input node n1 with output node n2. The gain parameter K is declared explicitly in the instantiation line.

```
.model tdk modfas VMAX=10 VMIN=-10 VSATP=9.9 VSATN=-9.9
...
ys4 expamp n1 n2 model:tdk
```

Specifies an expamp model with input node n1 and output node n2. Parameters VMAX, VMIN, VSATP and VSATN are declared using the **.MODEL** command.

Model Equations

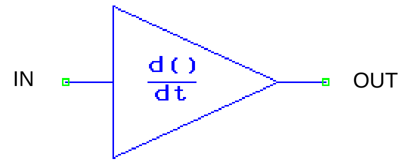
$$v(OUT) = LIMITER\{-K \times BASE^{v(IN)/E}\}$$

Information on voltage limiting is given in the "Voltage Limiter" on page 397. Non-limited voltages can be specified using `v(yname->lin)`.

Differentiator

```
Yxx DIFF [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-42. Differentiator Macromodel



The model provides the differentiated input signal at the output node.

Model Pins

- `IN`

  Name of the input node

- `OUT`

  Name of the output node

Table 6-25. Differentiator Model Parameters

Nr.	Name	Default	Units	Definition
1	K	1	V	Time constant
2	C0	1	V	DC value
3	SLR	1.0×10^9	V/s	Limits the slewrate of the signal on the `OUT` node
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
y1 diff pin: n1 n2 param: K=100.0
```

Specifies a diff model y1 having input node n1 with output node n2. The K parameter is declared explicitly in the instantiation line.

```
.model tdk modfas C0=-1.0
ys4 diff n1 n2 model:tdk
```

Specifies a diff model having input node n1 with output node n2. The C0 parameter is declared using the **.MODEL** command.

Model Equations

DC Analysis

$$v(OUT) = \mathrm{C0}$$

Transient Analysis

$$v(OUT) = -K \times \frac{d}{dt} v(IN)$$

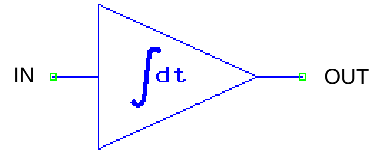
Frequency Analysis

$$\frac{v(OUT)}{v(IN)} = -j\omega K$$

Integrator

```
Yxx INTEG [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-43. Integrator Macromodel



This model provides the integrated input signal at the output node.

Model Pins

- `IN`

  Name of the input node

- `OUT`

  Name of the output node

Table 6-26. Integrator Model Parameters

Nr.	Name	Default	Units	Definition
1	K	1	V	Time constant
2	C0	1	V	DC value
3	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
y1 integ pin: n1 n2 param: K=100.0
```

Specifies an integ model y1 having input node n1 with output node n2. The K parameter is declared explicitly in the instantiation line.

```
.model tdk modfas C0=-1.0
ys4 integ n1 n2 model:tdk
```

Specifies an integ model having input node n1 with output node n2. The C0 parameter is declared using the **.MODEL** command.

Model Equations

DC Analysis

$$v(OUT) = C0$$

Transient Analysis

$$v(OUT) = \frac{-1}{K} \times \int v(IN)dt + C0$$

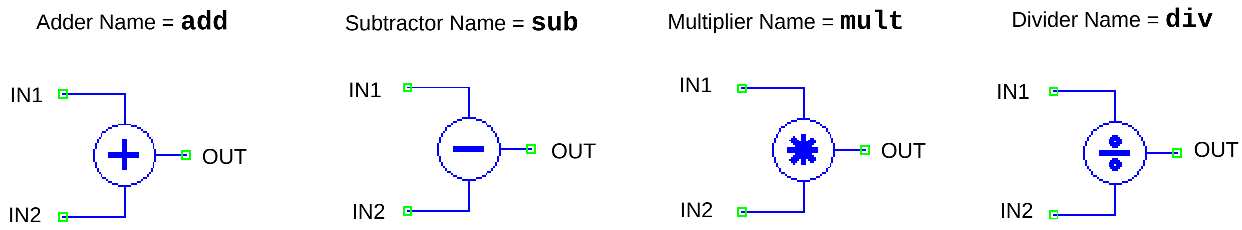
Frequency Analysis

$$\frac{v(OUT)}{v(IN)} = \frac{-1}{j\omega K}$$

Adder, Subtractor, Multiplier and Divider

```
Yxx NAME [PIN:] IN1 IN2 OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 6-44. Adder, Subtractor, Multiplier & Divider Macromodels



This model provides the specified arithmetic operation of the input signals at the output node.

NAME Parameter

- **ADD**

  Specifies the Adder model
- **SUB**

  Specifies the Subtractor model
- **MULT**

  Specifies the Multiplier model
- **DIV**

  Specifies the Divider model

Model Pins

- `IN1`

  Name of the first input node
- `IN2`

  Name of the second input node
- `OUT`

  Name of the output node

Table 6-27. Adder, Subtractor, Multiplier & Divider Model Parameters

Nr.	Name	Default	Units	Definition
1	VMAX	5	V	Maximum output voltage
2	VMIN	–5	V	Minimum output voltage
3	VSATP	4.75	V	Positive saturation voltage
4	VSATN	–4.75	V	Negative saturation voltage

Table 6-27. Adder, Subtractor, Multiplier & Divider Model Parameters

Nr.	Name	Default	Units	Definition
5	PSLOPE	1.0		Slope of Transfer Function at VSATP
6	NSLOPE	1.0		Slope of Transfer Function at VSATN
7	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Examples

```
y1 add pin: n1 n2 out param: VMAX=100.0
```

Specifies an adder model y1 with input nodes n1 and n2, and output node out. The VMAX parameter is declared explicitly in the instantiation line.

```
.model tdk modfas VMAX=10 VMIN=-10 VSATP=9.9 VSATN=-9.9
...
ys4 mult n1 n2 out model:tdk
```

Specifies a Multiplier model with input nodes n1 and n2, and output node out. The VMAX, VMIN, VSATP and VSATN parameters are declared using the **.MODEL** command.

Model Equations

$$v(OUT) = LIMITER\{v(\mathrm{IN1})[+|-|*|/]v(\mathrm{IN2})\}$$

Information on voltage limiting is given in the "Voltage Limiter" on page 397.

Chapter 7
Digital Macromodels

Eldo Digital Macromodels

The following digital macromodels are provided in Eldo:

Delay	DEL
Inverter	INV
Exclusive-OR Gate	XOR
2-Input Digital Gates	NAND, AND, NOR, OR
3-Input Digital Gates	NAND, AND, NOR, OR
Multiple Input Digital Gates	NAND, AND, NOR, OR

For ADC/DAC Mixed Signal Macromodels see "Mixed Signal Macromodels" on page 462.

Digital macromodels are implemented in Eldo as time variable voltage sources whose output values are computed at execution time.

These macromodels are parameterized with threshold voltages, speed and so forth. For digital gates, parameters can be specified through a **.MODEL** command as is the case for semiconductor devices. Values specified in the macromodel instantiation line supersede values specified in the **.MODEL** command.

For more information on the **.MODEL** command, see "Digital Model Definition" on page 449.

DYND2ALOG

The option **DYND2ALOG** can be used to dynamically calculate the threshold values VTHI and VTLO for digital macromodels using actual values of the high and low bias. The option works by taking the values from two extra pins defined by the user in the macromodel instantiation line. The value specified on the first pin provides the high voltage digital output value (VHI) and the value specified on the second pin provides the low voltage digital output value (VLO). The active threshold values **VTHI** and **VTLO**, will be computed dynamically using the following equations:

```
VTHI_ACTIVE = VLO + VTHI*DV
VTLO_ACTIVE = VLO + VTLO*DV
```

VTHI and VTLO are the values specified in the **.MODEL** command defining the digital macromodel or in the macromodel instance, VLO is the low output voltage value given on the second extra pin defined by the user, and DV is the voltage difference given by VHI - VLO.

For more information on the **DYND2ALOG** option, see "DYND2ALOG" on page 1018.

Digital Model Definition

```
.MODEL MNAME LOGIC [VHI=VAL] [VLO=VAL] [VTH=VAL]
+ [VTHI=VAL] [VTLO=VAL] [TPD=VAL] [TPDUP=VAL]
+ [TPDOWN=VAL] [CIN=VAL] [DRVL=VAL] [DRVH=VAL]
```

Used for the definition of digital gate models.

Parameters

- MNAME

  Name of the model

- **LOGIC**

  Defines the model as a digital gate model

- **VHI**=VAL

  Output voltage for the 1 logical state. Default value is 5V.

- **VLO**=VAL

  Output voltage for the 0 logical state. Default value is 0V.

- **VTH**=VAL

  Threshold input voltage. Default value is 2.5V.

- **VTHI**=VAL

  Input threshold voltage for the rising edge

- **VTLO**=VAL

  Input threshold voltage for the falling edge

Note

If only **VTH** is specified, then **VTHI**=**VTLO**=**VTH**. If both **VTHI** and **VTLO** are specified, **VTH** will be ignored. Ensure that the DC operating point is either above **VTHI** or below **VTLO**, otherwise the output may behave unpredictably in the first few cycles of simulation. **VTHI** and **VTLO** can be computed dynamically using the option **DYND2ALOG**.

- **TPD**=VAL

  Transit time through the gate (time from input threshold intersection to output threshold intersection). Default value is 1ns.

- **TPDUP**=VAL

  Transit time (See above) for output to reach **VTHI** volts (a value causing a change in the next gate)

- **TPDOWN**=VAL

  Transit time (See above) for output to reach **VTLO** volts (a value causing a change in the next gate)

Note

If only **TPD** is specified then **TPDUP**=**TPDOWN**=**TPD**. If both **TPDUP** and **TPDOWN** are specified, **TPD** will be ignored. The slopes at the output are determined by the values of **TPD**, **TPDUP** and **TPDOWN** (See Figure 7-1 on page 450).

The above definitions of transit times and threshold voltages make sure that transit times are additive through a chain of digital operators. It is not realistic, however, to model an Eldo digital gate as a chain of single gates, as the transition at the output of such a gate would not occur immediately.

- **CIN**=VAL

  Capacitance seen at one of the macro inputs, modeling the interconnection capacitance. All inputs are loaded with the same capacitance. **CIN** has no effect when the macros are linked in a chain, as the input of one gate is the output of another, which is modeled as a voltage source. Default value is zero.

- **DRVL**=VAL

  Drive resistance for the logic 0 state

- **DRVH**=VAL

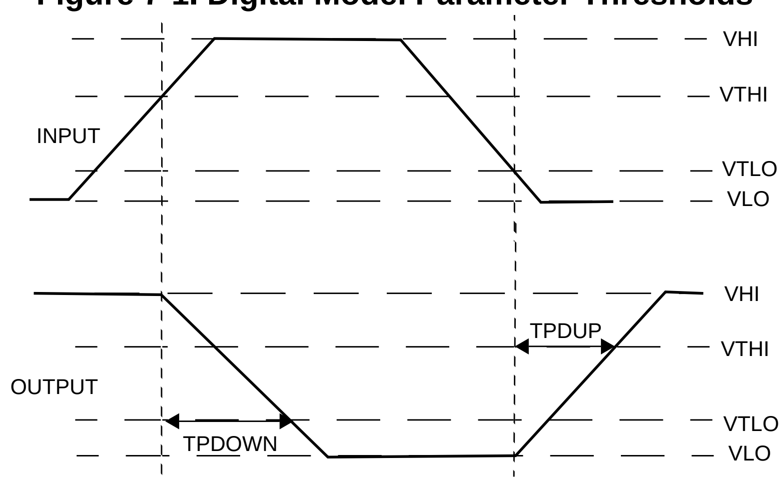
  Drive resistance for the logic 1 state

Note

These drive resistances are only relevant when current source modeling of Eldo logic primitive outputs is used, as selected using **.OPTION ULOGIC**. By default, if **.OPTION ULOGIC** is not specified, a simple voltage source is used to model logic primitive outputs.

See the "ULOGIC" on page 983.

Figure 7-1. Digital Model Parameter Thresholds



Example

```
.model nand_1 logic vlo=5 vlo=-5 vth=0
...
nand#a n1 n2 o1 nand_1 tpd=2.5n cin=0.5p
```

Specifies the two input nand gate nand#a of model type nand_1. The parameters of the gate are described using the **.MODEL ... LOGIC** command and indicate that the voltages used for the logical states 0 (vlo) and 1 (vlo) are 5V and 5V respectively and that the threshold input voltage (vth) is set to zero. The time for the output to reach vth is 2.5ns and the input capacitance is 0.5pF.

The example below shows how the **DYND2ALOG** option is used to compute dynamic values for VHI and VLO.

```
*DYND2ALOG example
.option dynd2alog
.SUBCKT A2_020 A A0 A1 BIAS VN VP
    AND2UX01 A0 A1 A vp vn 0020
    .MODEL 0020 LOGIC
*   + VHI    = 5
*   + VLO    = 0
    + VTHI   = (2/3)
    + VTLO   = (1/3)
    + TPDUP  = 1ns
    + TPDOWN = 1ns
    + CIN    = 0.05pF
.ENDS A2_020

vn vn 0 1
vp vp 0 3
va a 0 pwl ( 0 0 10n 0 15n 5)
vb b 0 pwl ( 0 0 5n 5 20n 5 25n 0)
x1 out a b 0 vn vp a2_020
x2 out2 out a 0 vn vp a2_020
.tran 1n 100n
.plot tran v(a)
.plot tran v(b)
.plot tran v(out)
.plot tran v(out2)
.extract tran yval(v(out),20n)
.extract tran yval(v(out),10n)
.end
```

A 2-input AND gate is instantiated using the model defined in the **.MODEL** statement. In the instantiation line two extra pins are defined as `VP` and `VN`. These are used by the option **DYND2ALOG** which has been defined at the start of the netlist. The difference between calculating the values for VHI and VLO dynamically and declaring them in the **.MODEL** statement can be seen using the two **.EXTRACT** commands defined in the netlist. These statements will extract the values on the output waveform v(out) for x-axis values 20ns and 10ns. The x-axis values define the time when the output voltage is at its high and low states, therefore correspond to VHI and VLO.

Running the simulation with the **DYND2ALOG** option specified and including the two extra pins will produce the values 3V and 1V for VHI and VLO respectively. Removing the option and the pins and including the VHI and VLO parameters in the **.MODEL** statement will produce the values specified on these parameters, in this case they are 5V and 0V.

Delay

```
DELxx IN OUT VAL
```

Figure 7-2. Delay Macromodel

This macromodel describes an ideal delay element that transfers its input voltage to its output after a specified time delay, where the reference node is ground. The input impedance is infinite.

Parameters

- `IN`

  Name of the input node
- `OUT`

  Name of the output node
- `xx`

  Delay element identifier (ASCII string)
- `VAL`

  Value of the delay in seconds

Example

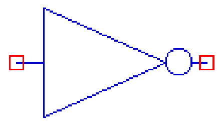
```
del1 a1 a2 2.0e-9
```

Specifies a delay element placed between nodes `a1` and `a2`, with a delay of 2ns.

Inverter

```
INVxx IN OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Figure 7-3. Inverter Macromodel



Parameters

- `IN`

  Name of the input node

- `OUT`

  Name of the output node

- `xx`

  Inverter element identifier (ASCII string)

- `REF1 REF2`

  Only to be used when specifying **`.OPTION DYND2ALOG`**. Names of the pins used in the dynamic calculation of the threshold values. See "DYND2ALOG" on page 447 for more details.

- `MNAME`

  Name of a model described with the **`.MODEL`** command

- `PAR=VAL`

  A direct assignment of a **`.MODEL`** command parameter

Example

```
.model inv logic vhi=5 vlo=-5 vthi=1.0 vtlo=1.0
+ tpd=2.5n cin=0.5p
...
inv44 i1 o1 inv
```

Specifies the inverter inv44 of model type inv placed between the nodes i1 and o1. The parameters of the inverter are specified using the **`.MODEL`** command.

```
inv44 i1 o1 vhi=5 vlo=-5 vthi=1.0 vtlo=1.0
+ tpd=2.5n cin=0.5p
```

Specifies the same inverter as above but with its parameters assigned directly instead of using the **`.MODEL`** command.

For more information, refer to the “.MODEL” on page 723.

Exclusive-OR Gate

```
XORxx IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Parameters

- `IN1, IN2`

  Names of the input nodes

- `OUT`

  Name of the otput node

- `xx`

  Exclusive-OR element identifier (ASCII string)

- `REF1 REF2`

  Only to be used when specifying **`.OPTION DYND2ALOG`**. Names of the pins used in the dynamic calculation of the threshold values. See "DYND2ALOG" on page 447 for more details.

- `MNAME`

  Name of a model described with the **`.MODEL`** command

- `PAR=VAL`

  A direct assignment of a **`.MODEL`** command parameter

Example

```
.model xor logic vhi=5 vlo=-5 vthi=1.0 vtlo=1.0
+ tpd=2.5n cin=0.5p
...
xor44 i1 i2 o1 xor
```

Specifies the exclusive-OR gate xor44 of model type xor placed between the nodes i1, i2 and o1. The parameters of the exclusive-OR are specified using the **`.MODEL`** command.

```
xor44 i1 i2 o1 vhi=5 vlo=-5 vthi=1.0 vtlo=1.0
+ tpd=2.5n cin=0.5p
```

Specifies the same exclusive-OR as above but with its parameters assigned directly instead of using the **`.MODEL`** command.

For more information, refer to the ".MODEL" on page 723.

2-Input Digital Gates

```
DGATExx IN1 IN2 OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Parameters

- `DGATE`

  Digital gate type

Table 7-1. 2-Input Digital Gate Types

Gate Type	Function
NAND	NAND gate
AND	AND gate
NOR	NOR gate
OR	OR gate

- `xx`

  Digital gate identifier (ASCII string)

Note

The first ASCII character of the gate identifier (xx) must not be a 3 since this would indicate a triple-input gate.

- `IN1, IN2`

  Names of the input nodes

- `OUT`

  Name of the output node

- `REF1 REF2`

  Only to be used when specifying **.OPTION DYND2ALOG**. Names of the pins used in the dynamic calculation of the threshold values. See “DYND2ALOG” on page 447 for more details.

- `MNAME`

  Name of a model described with the **.MODEL** command

- `PAR=VAL`

  A direct assignment of a **.MODEL** command parameter

Examples

```
*.MODEL definition
.model nand_1 logic vhi=5 vlo=-5 vth=0 tpd=2.5n cin=0.5p
...
nand4 n1 n2 o1 nand_1
```

Specifies a two input NAND gate nand4 of model type nand_1 with input nodes n1 and n2 and output node o1. The model parameters of the NAND gate are described using the **.MODEL** command.

```
nand4 n1 n2 o1 vhi=5 vlo=-5 vth=0 tpd=2.5n cin=0.5p
```

Specifies the same NAND gate as above but with its parameters assigned directly instead of with the **.MODEL** command.

For more information, refer to the “.MODEL” on page 723.

3-Input Digital Gates

```
DGATExx IN1 IN2 IN3 OUT [REF1 REF2] [MNAME] [PAR=VAL]
```

Parameters

- `DGATE`

  Digital gate type

Table 7-2. 3-Input Digital Gate Types

Gate Type	Function
NAND3	NAND gate
AND3	AND gate
NOR3	NOR gate
OR3	OR gate

- `xx`

  Digital gate identifier (ASCII string)

- `IN1, IN2, IN3`

  Names of the input nodes

- `OUT`

  Name of the output node

- `REF1 REF2`

  Only to be used when specifying **`.OPTION DYND2ALOG`**. Names of the pins used in the dynamic calculation of the threshold values. See "DYND2ALOG" on page 447 for more details.

- `MNAME`

  Name of a model described with the **`.MODEL`** command

- `PAR=VAL`

  A direct assignment of a **`.MODEL`** command parameter

Examples

```
*AND3 .MODEL definition
.model and_1 logic vhi=5 vlo=-5 vth=0 tpd=2.5n cin=0.5p
...
*main circuit
and3_1 n1 n2 n3 o1 and_1
```

Specifies a three input AND gate and3_1 with input nodes n1, n2 and n3 and output node o1. The parameters of the and gate are described in the model and_1 using the **`.MODEL`** command.

```
and3_1 n1 n2 n3 o1 vhi=5 vlo=-5 vth=0 tpd=2.5n cin=0.5p
```

Specifies the same AND gate as above but with its parameters assigned directly instead of with the **.MODEL** command.

For more information, refer to “.MODEL” on page 723.

Multiple Input Digital Gates

```
DGATExx IN1 IN2...{INX} OUT [REF1 REF2] MNAME [PAR=VAL]
```

Multiple Input Digital Gates Macromodel.

Parameters

- `DGATE`

  Digital gate type

Table 7-3. Multiple Input Digital Gate Types

Gate Type	Function
NAND#	NAND gate
AND#	AND gate
NOR#	NOR gate
OR#	OR gate

- `xx`

  Digital gate identifier (ASCII string)

- `IN1, IN2,...{INX}`

  Names of the input nodes

- `OUT`

  Name of the output node

- `REF1 REF2`

  Only to be used when specifying **.OPTION DYND2ALOG**. Names of the pins used in the dynamic calculation of the threshold values. See "DYND2ALOG" on page 447 for more details.

- `MNAME`

  Name of a model described with the **.MODEL** command

- `PAR=VAL`

  A direct assignment of a **.MODEL** command parameter

Examples

```
*AND# .MODEL definition
.model and_1 logic vhi=5 vlo=-5 vth=0 tpd=2.5n cin=0.5p
...
*main circuit
and#_1 n1 n2 n3 n4 o1 and_1
```

Specifies an AND gate and#_1 with four input nodes n1, n2, n3 and n4 and an output node o1. The parameters of the AND gate are described in the model and_1 using the **.MODEL** command.

```
and#_1 n1 n2 n3 n4 o1 and_1 vhi=5 vlo=0 vth=2.5
+ tpd=2.5n cin=0.5p
```

In this example, the parameters on the instantiation line override the parameters in the **.MODEL** command.

For more information, refer to ".MODEL" on page 723.

Mixed Signal Macromodels

The following mixed signal macromodels are provided in Eldo:

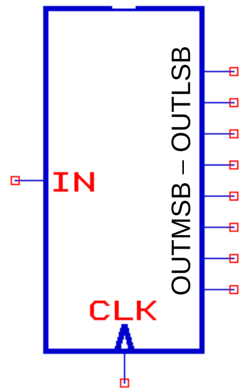
Analog to Digital Converter	ADC
Digital to Analog Converter	DAC

For Digital Macromodels, see "Eldo Digital Macromodels" on page 447.

Analog to Digital Converter

```
ADCxx CLK IN OUTSB{OUTSB} [EDGE=VAL] [VTH=VAL] [VHI=VAL]
+ [VLO=VAL] [VINF=VAL] [VSUP=VAL] [TCOM=VAL] [TPD=VAL]
```

Figure 7-4. Analog to Digital Converter Macromodel



The analog to digital converter (ADC) is defined by the clock, the analog input and a number of digital outputs. Outputs are computed only when the clock validates the input, i.e. on the rising or falling edge depending on the value of the **EDGE** parameter.

Parameters

- xx

  Analog to digital converter identifier (ASCII string)

- CLK

  Name of the clock node

- IN

  Name of the analog input node

- OUTSB

  Digital output nodes (MSB to LSB). A maximum of 31 bits can be defined when using this macromodel.

- **EDGE**=VAL

  EDGE=1 to validate the output on the rising edge of the clock

  EDGE=-1 to validate the output on the falling edge of the clock. Default value is 1.

- **VTH**=VAL

  Threshold voltage for the clock. Default value is 2.5V.

- **VHI**=VAL

  Voltage corresponding to the 1 output logical state. Default value is 5V.

- **VLO**=VAL

  Voltage corresponding to the 0 output logical state. Default value is 0V.

- **VINF**=VAL

  Lower input voltage. If an analog voltage entering the ADC is lower than this value, all outputs remain at **VLO**. Default value is 0V.

- **VSUP**=VAL

  Upper input voltage. If an analog voltage entering the ADC is higher than this value, all outputs remain at **VHI**. Default value is 5V.

- **TCOM**=VAL

  Time for the outputs to change from **VHI** to **VLO** or vice versa. Default value is 1ns.

- **TPD**=VAL

  Transit time through the converter. The output will start changing **TPD** seconds after the clock validates the input. Default value is 10ns.

Example

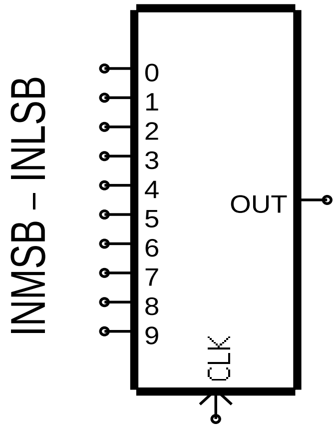
```
adc_1 clk in d4 d3 d2 d1 edge=-1 vth=1 vhi=5
+ vlo=0 vinf=1.0 vsup=4.0 tcom=5n tpd=2n
```

Specifies an ADC named adc_1 with clock node clk, analog input node in and digital output nodes d4 (MSB), d3, d2 and d1 (LSB). The output is validated on the falling edge of the clock, with the threshold voltage for the clock being 1V. The voltages corresponding to logical 1 and 0 are 5V and 0V respectively. The upper and lower threshold voltages in order for the output to remain high or low are 4V and 1V respectively with the time for the ADC to change from a high to a low voltage being 5ns. Finally, the transit time through the ADC is 2ns.

Digital to Analog Converter

```
DACxx CLK INSB{INSB} OUT [EDGE=VAL] [VTH=VAL] [VTIN=VAL]
+ [VHI=VAL] [VLO=VAL] [TPD=VAL] [SL=VAL]
```

Figure 7-5. Digital to Analog Converter Macromodel



The digital to analog converter (DAC) is defined by the clock, the digital inputs and the analog output. The output is computed only when the clock validates the input, i.e. on the rising or falling edge depending on the value of the **EDGE** parameter.

Parameters

- `xx`

  Digital to analog converter identifier (ASCII string)

- `CLK`

  Name of the clock node

- `INSB`

  Name of the digital input nodes (MSB to LSB)

- `OUT`

  Name of the analog output node. A maximum of 31 bits can be defined when using this macromodel.

- **EDGE**=VAL

  EDGE=1 validates the output on the rising edge of the clock

  EDGE=-1 validates the output on the falling edge of the clock. Default value is 1.

- **VTH**=VAL

  Threshold voltage for the clock. Default value is 2.5V.

- **VTIN**=VAL

  Threshold voltage for the inputs. Default value is 2.5V.

- **VHI**=VAL

  Voltage output when all inputs are above **VTIN**. Default value is 5V.

- **VLO**=VAL

  Voltage output when all inputs are below **VTIN**. Default value is 0V.

- **TPD**=VAL

  Transit time through the converter. The output will start changing **TPD** seconds after the clock validates the output. Default value is 10ns.

- **SL**=VAL

  Slope at the output, in Vs^{-1}. Default value is 0.1×10^{9} Vs^{-1} (0.1 Vns^{-1}).

Example

```
dac2 clk s4 s3 s2 s1 out vth=2 vlo=1 tpd=5n sl=1e9
+ vtin=2.2 vth=2.5
```

Specifies a DAC named dac2 with clock node clk, digital inputs s4 (MSB), s3, s2 and s1 (LSB) and analog output out. The threshold voltages for the clock and inputs are 2.5V and 2.2V respectively. When all inputs are above the input threshold voltage, the output voltage is equal to 2V, and when they are all below it the output voltage is equal to 1V. Finally, the slope of the output is defined as 1Vns^{-1}.

Chapter 8
Magnetic Macromodels

Eldo Magnetic Macromodels

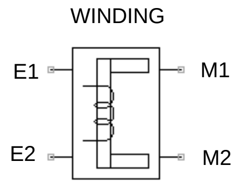
The following magnetic macromodels are provided in Eldo:

Transformer Winding	WINDING
Non-linear Magnetic Core 1	NLCORE1
Non-linear Magnetic Core 2	NLCORE2
Linear Magnetic Core	LINCORE
Magnetic Air Gap	AIRGAP
Transformer (Variable # of Windings)	LVTRANS
Ideal Transformer	JTRAN

Transformer Winding

```
Yxx WINDING [PIN:] E1 E2 M1 M2 [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 8-1. Transformer Winding Macromodel



This is a macromodel for a winding describing the interaction between the electrical and magnetic domain of a wire wrapped around a linear/non-linear material.

Model Pins

- `E1`

  Name of the first electrical pin
- `E2`

  Name of the second electrical pin
- `M1`

  Name of the first magnetic pin
- `M2`

  Name of the second magnetic pin

Table 8-1. Transformer Winding Model Parameters

Nr.	Name	Default	Units	Definition
1	N	1.0×10^{3}		Number of turns
2	R	1.0×10^{-2}	Ω	Resistance of the winding
3	K	1.0		Coupling coefficient of the winding to the core. May be in the range $0.0 \leq K \leq 1.0$
4	M[1]	1		Device multiplier

1. **`.OPTION YMFACT`** must be specified for **`M`** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Example

```
ymod5 winding n1 n2 p1 p2
```

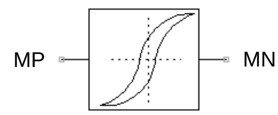
Specifies a winding ymod5 of type winding having electrical pins n1 and n2 with magnetic pins p1 and p2. Default model parameters are used.

Non-linear Magnetic Core 1

```
Yxx NLCORE1 [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 8-2. Non-linear Magnetic Core 1 Macromodel

NLCORE1



This is a macromodel for a non-linear magnetic core. It is a physically based mathematical model of the ferromagnetic hysteresis, which includes the following effects:

- Mean field approach for domain coupling.
- Domain wall motion.
- Pinning of domain walls on defect sites.
- Frequency dependent domain wall pinning.

The source information for this macromodel can be found in the following technical articles:
D.C. Jiles, D.L. Atherton, "*Theory of Ferromagnetic Hysteresis*"
"Journal of Magnetism and Magnetic Materials," Vol. 61, Sept. 1986, pp 48-60.
R. Brachtendorf, R. Laur, "*Modeling of Magnetic Elements including Frequency Effects*."
2.GME/ITG Workshop "*Entwicklung von Analogschaltungen mit CAE-Methoden*."
Ilmenau, Germany, March 1993.

Model Pins

- `MP`

  Name of the input magnetic node
- `MN`

  Name of the output magnetic node

Table 8-2. Non-linear Magnetic Core 1 Model Parameters

Nr.	Name	Default	Units	Definition
1	AREA	1.0×10^{-4}	m^2	Core area
2	LEN	1.0×10^{-3}	m	Length of the magnetic path
3	MS	1.7×10^{6}	Am^{-1}	Saturation magnetization
4	ALPHA	1.0×10^{-3}		Domain coupling coefficient (mean field parameter)

Table 8-2. Non-linear Magnetic Core 1 Model Parameters

Nr.	Name	Default	Units	Definition
5	A	1.0×10^{3}	Am^{-1}	Domain density
6	K	1.0×10^{3}	Am^{-1}	Domain wall pinning coefficient
7	C	0.1		Reversible wall motion coefficient
8	KF	1.0×10^{-6}		Frequency dependent domain wall pinning coefficient
9	LEVEL	1		Selector for Anhysterisis model LEVEL=1—Langevin function LEVEL=2—Tangens-Hyperbolicus (tanh)
10	MD	1.0×10^{-5}		Delay element for irreversible magnetization
11	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See "General Notes on the Use of FAS Macromodels" on page 380 for additional notes on using this model.

Example

```
ymod1 nlcore1 n1 n2
```

Specifies a non-linear core ymod1 of type nlcore1 having input magnetic node n1 and output magnetic node n2. Default model parameters are used.

Model Characteristics

Typical hysteresis curves for this macromodel are shown on the following page:

Figure 8-3. Symmetric B-H loops with Different Amplitudes

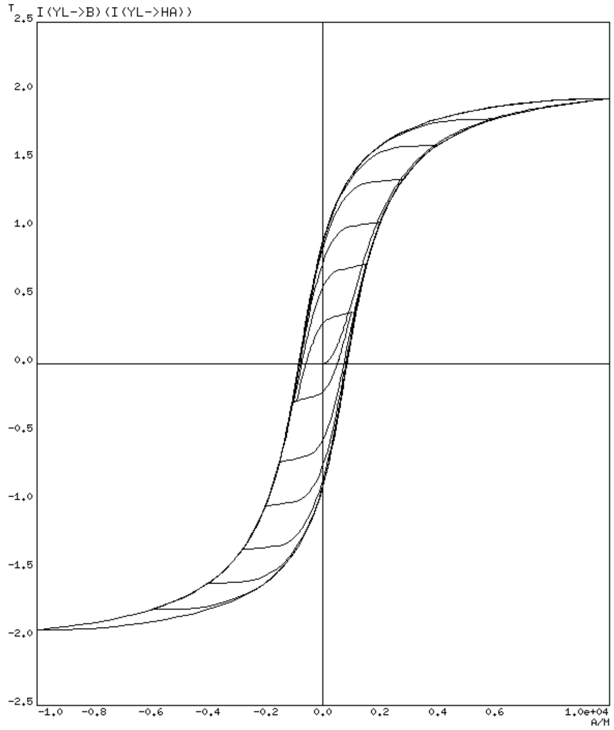
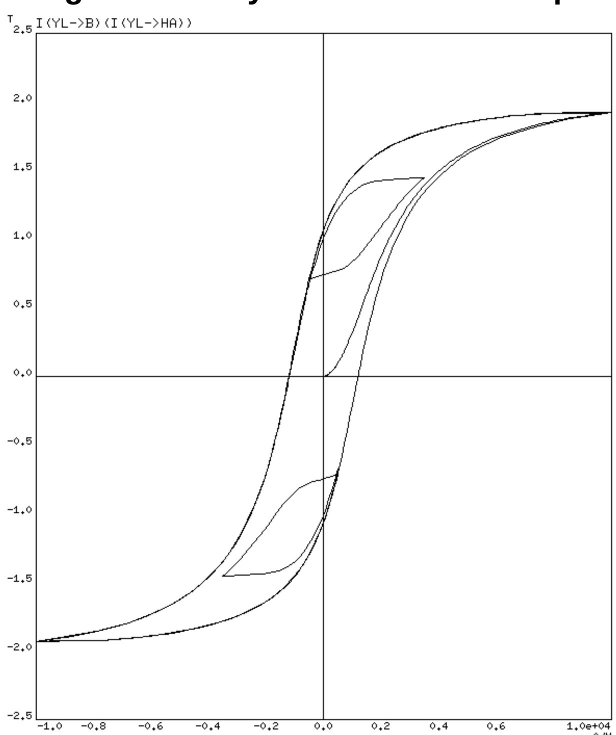


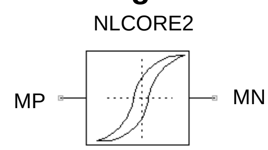
Figure 8-4. Asymmetric Minor Loops



Non-linear Magnetic Core 2

```
Yxx NLCORE2 [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 8-5. Non-linear Magnetic Core 2 Macromodel



A non-linear magnetic core macromodel. It describes the hysteretic behavior of a magnetic material including the temperature and frequency dependence of the hysteresis characteristic.

The source information for this macromodel can be found in the following article: Chan, Vladimirescu, Gao, Liebmann, Valainis, "*Non-linear Transformer Model for Circuit Simulation*" 'IEEE Transactions on Computer-Aided Design,' Vol. 10, No. 4, April 1991.

Model Pins

- `MP`

  Name of the input magnetic node

- `MN`

  Name of the output magnetic node

Table 8-3. Non-linear Magnetic Core 2

Nr.	Name	Default	Units	Definition
1	AREA	1.0×10^{-4}	m^2	Core area
2	LEN	5.0×10^{-2}	m	Length of the magnetic path
3	HC	10.0	Am^{-1}	Coercive magnetic field strength
4	BR	0.1	Vsm^{-2}	Remnant magnetic flux density
5	BS	1.0	Vsm^{-2}	Saturation magnetic flux density
6	CEPS	1.0×10^{-2}		Coefficient influencing the internal model accuracy
7	TBS	0.0	K^{-1}	Temperature coefficient for BS
8	TBR	0.0	K^{-1}	Temperature coefficient for BR
9	THC	0.0	K^{-1}	Temperature coefficient for HC
10	FNOM	1.0×10^{3}	Hz	Working frequency
11	FC1	1.0		First frequency coefficient

Table 8-3. Non-linear Magnetic Core 2

Nr.	Name	Default	Units	Definition
12	FC2	0.0	Hz^{-1}	Second frequency coefficient
13	FC3	0.0		Third frequency coefficient
14	MYI	1000.0		Initial relative permeability of the core material
15	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Examples

```
ymod1 nlcore2 n1 n2
```

Specifies a non-linear magnetic core ymod1 of type nlcore2 having input magnetic node n1 and output magnetic node n2. Default model parameters are used.

```
.model mod modfas bs=0.5 br=0.2 hc=20.0
...
ycore1 nlcore2 n1 n2 model: mod
```

Specifies a non-linear magnetic core ycore1 of type nlcore2 with input magnetic node n1 and output magnetic node n2. Characteristic points of the hysteresis curve are declared using the **.MODEL** command.

Model Characteristics

Typical hysteresis curves for this macromodel are shown on the following page:

Figure 8-6. Symmetric B-H loops with Different Amplitudes

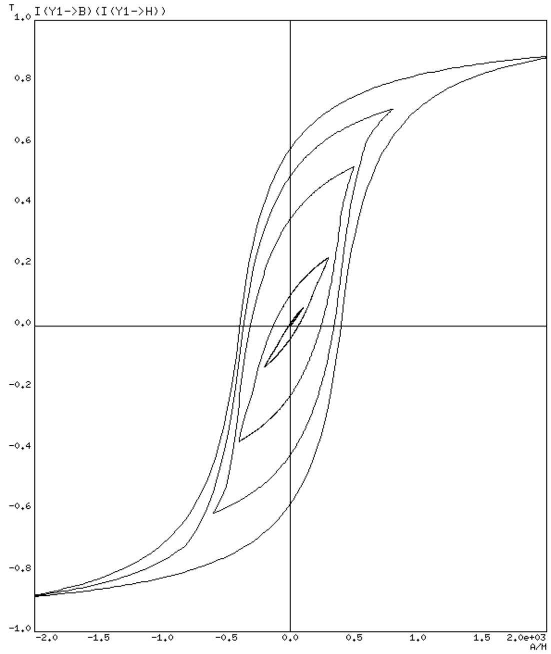
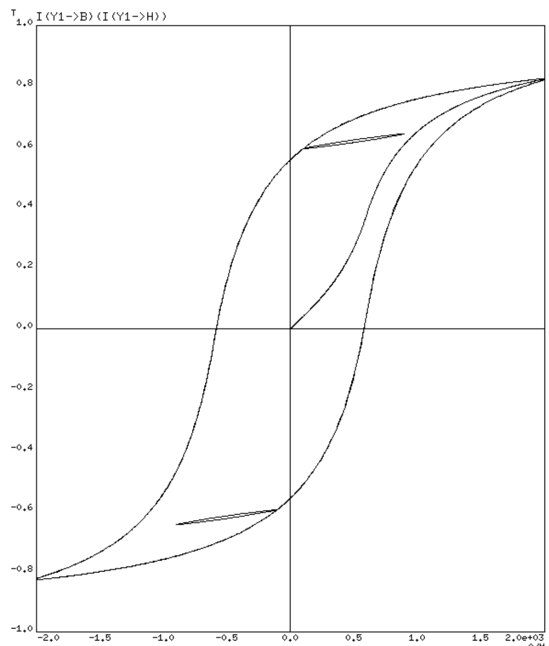


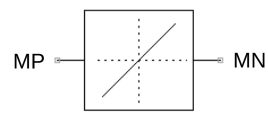
Figure 8-7. Asymmetric Minor Loops



Linear Magnetic Core

```
Yxx LINCORE [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 8-8. Linear Magnetic Core Macromodel



This is a macromodel for a magnetic core with linear B-H characteristics.

Model Pins

- `MP`

  Name of the input magnetic node

- `MN`

  Name of the output magnetic node

Table 8-4. Linear Magnetic Core Model Parameters

Nr.	Name	Default	Units	Definition
1	AREA	1.0×10^{-4}	m^2	Core area
2	LEN	1.0×10^{-2}	m	Length of the magnetic path
3	MYR	1		Relative permeability of core material
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Example

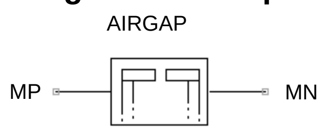
```
ymod1 lincore n1 n2
```

Specifies a linear core ymod1 of type lincore having input magnetic node n1 and output magnetic node n2. Default model parameters are used.

Magnetic Air Gap

```
Yxx AIRGAP [PIN:] MP MN [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 8-9. Magnetic Air Gap Macromodel



A linear resistor macromodel modeling the magnetic resistance of an air gap inside a magnetic core.

Model Pins

- `MP`

  Name of the input magnetic node

- `MN`

  Name of the output magnetic node

Table 8-5. Magnetic Air Gap Model Parameters

Nr.	Name	Default	Units	Definition
1	AGAP	1.0×10^{-4}	m^2	Cross-sectional area of the air gap
2	LGAP	1.0×10^{-2}	m	Length of the air gap
3	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

See “General Notes on the Use of FAS Macromodels” on page 380 for additional notes on using this model.

Example

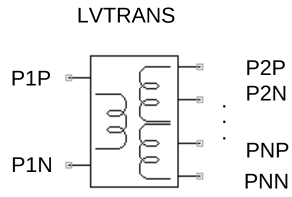
```
ymod1 airgap n1 n2
```

Specifies a magnetic air gap ymod1 of type airgap having input magnetic node n1 and output magnetic node n2. Default model parameters are used.

Transformer (Variable # of Windings)

```
Yxx LVTRANS [PIN:] P1P P1N P2P P2N {PNP PNN}
+ [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 8-10. Transformer (Variable # of Windings) Macromodel



A macromodel for a linear transformer with a variable number (maximum is 8) of windings. The number of pins at instantiation determines the number of transformer windings and hence the number of required parameters.

Model Pins

- `PNP`

  Name of the positive pin of the nth transformer winding (dependent on the number of transformer windings declared).

- `PNN`

  Name of the negative pin of the nth transformer winding (dependent on the number of transformer windings declared).

Table 8-6. Transformer Model Parameters

Nr.	Name	Default	Units	Definition
1	L_{ij}	1.0×10^{-3}	H	Element ij of inductance matrix
2	R_i	1.0×10^{-3}	Ω	Element i of winding resistance vector
3	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Where i=1 to number of windings, j=1 to number of windings.

Example 1—Transformer with 2 Windings

```
ytr1 lvtrans p1 p2 s1 s2 param: l11=2.0e-3 l12=1.0e-3
+ l21=2.0e-3 l22=2.0e-3 r1=1 r2=10
```

Specifies a transformer ytr1 of type lvtrans with two windings. The first transformer winding has pins p1 (+ve) and p2 (-ve) and the second winding has pins s1 (+ve) and s2 (-ve). The model equations given below show the inductance matrix and the resistance vector for the above transformer.

$$\begin{bmatrix} v(\mathrm{p1,p2}) \\ v(\mathrm{s1,s2}) \end{bmatrix} = \begin{bmatrix} \mathrm{L11} & \mathrm{L21} \\ \mathrm{L12} & \mathrm{L22} \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}\mathrm{i1} \\ \frac{d}{dt}\mathrm{i2} \end{bmatrix} + \begin{bmatrix} \mathrm{R1} & 0 \\ 0 & \mathrm{R2} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{i1} \\ \mathrm{i2} \end{bmatrix}$$

where ***i1*** is the current in the first winding and ***i2*** is the current in the second winding.

Example 2—Transformer with 3 Windings

```
ytr3 lvtrans p1 p2 s1 s2 t1 t2
```

Specifies a transformer ytr3 of type lvtrans with three windings. The first transformer winding has pins p1 (+ve) and p2 (-ve), the second winding has pins s1 (+ve) and s2 (-ve) and the third winding has pins t1 (+ve) and t2 (-ve). Default parameters are used. The model equations given below show the inductance matrix and the resistance vector for the above transformer.

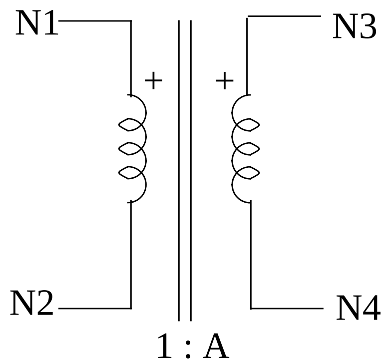
$$\begin{bmatrix} V(\mathrm{p1,p2}) \\ V(\mathrm{s1,s2}) \\ V(\mathrm{t1,t2}) \end{bmatrix} = \begin{bmatrix} \mathrm{L11} & \mathrm{L21} & \mathrm{L31} \\ \mathrm{L12} & \mathrm{L22} & \mathrm{L32} \\ \mathrm{L13} & \mathrm{L23} & \mathrm{L33} \end{bmatrix} \cdot \begin{bmatrix} \frac{d}{dt}\mathrm{i1} \\ \frac{d}{dt}\mathrm{i2} \\ \frac{d}{dt}\mathrm{i3} \end{bmatrix} + \begin{bmatrix} \mathrm{R1} \\ \mathrm{R2} \\ \mathrm{R3} \end{bmatrix} \cdot \begin{bmatrix} \mathrm{i1} \\ \mathrm{i2} \\ \mathrm{i3} \end{bmatrix}$$

where ***i1*** is the current flowing in the first winding, ***i2*** is the current in the second winding and ***i3*** is the current in the third winding.

Ideal Transformer

```
Yxx JTRAN N1 N2 N3 N4 [PARAM: A=VAL]
```

Figure 8-11. Ideal Transformer Macromodel



A macromodel for an ideal transformer.

Parameters

- `N1, N2, N3, N4`

  Ideal transformer nodes as shown in the figure above

- `A`

  Transformer's turns ratio

Example

AC simulation of an ideal transformer (1:2).

```
.param tran_ratio=2

Ytran jtran ain 0 aout 0 PARAM: a=tran_ratio

Vin      Xin     0     AC  10
Vdummy1  Xin     ain   AC  0
Vdummy2  aout    Xout  AC  0
Rout     Xout    0     1

.defwave voltage_gain=V(aout)/V(ain)
.defwave current_gain=I(Vdummy2)/I(Vdummy1)
.ac dec 10 1 1g

.plot ac wm(voltage_gain)
.plot ac wp(voltage_gain)
.plot ac wm(current_gain)
.plot ac wp(current_gain)

.end
```

Chapter 9
Switched Capacitor Macromodels

Introduction

This chapter is concerned with the area of Switched Capacitor Macromodels. Within the subject area of Switched Capacitor Macromodels, there are two separate types to be distinguished between and they are:

- Switch Level Representation
- Z-domain Representation

Switch Level Representation

The following macromodels have a Charge Conservation characteristic and are the key elements for Switched Capacitor (SC) applications, where the MOS analog switches are represented by an approximate general linearized model of a non-ideal switch.

All these models are Kirchhoff type elements and can be applied as normal components in an electronic circuit in the same way as other models such as bipolar transistors, diodes, noise sources, and so on.

All switched capacitor networks built using these models can be analyzed in the time domain without any restrictions, but an AC analysis would be meaningless. If an AC analysis is needed, particularly for switched capacitor filter applications, then another type of model representation for the switched capacitor circuit is required.

Refer to "Z-domain Representation" on page 491.

Macromodels

Below is a list of the Eldo Macromodels which are described throughout this chapter:

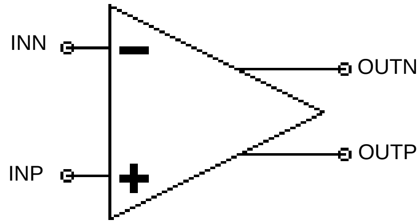
Operational Amplifier	**OPA**xx
Switch	**S**xx
Ideal Operational Amplifier	**Y**xx **SC_IDEAL**
Inverting Switched Capacitor	**Y**xx **SC_I**

Non-inverting Switched Capacitor	**Y**xx **SC_N**
Parallel Switched Capacitor	**Y**xx **SC_P**
Serial Switched Capacitor	**Y**xx **SC_S1** **Y**xx **SC_S2**
Serial-parallel Switched Capacitor	**Y**xx **SC_SP1** **Y**xx **SC_SP2**
Bi-linear Switched Capacitor	**Y**xx **SC_B**
Unswitched Capacitor	**Y**xx **SC_U**

Operational Amplifier

```
OPAxx INP INN OUTP OUTN [MNAME] [LEVEL=VAL] [VOFF=VAL]
+ [SL=VAL] [CIN=VAL] [RS=VAL] [VSAT=VAL] [VSATM=VAL]
+ [GAIN=VAL] [FC=VAL] [FNDP=VAL] [IMAX=VAL] [CMRR=VAL]
```

Figure 9-1. Operational Amplifier Macromodel



A macromodel for single- and two-stage operational amplifiers. This syntax supersedes that of all previous Eldo versions. Old syntax is still supported, but no longer recommended.

Parameters

- `xx`

  Amplifier name
- `INP`

  Name of the positive input node
- `INN`

  Name of the negative input node
- `OUTP`

  Name of the positive output node
- `OUTN`

  Name of the negative output node. For single-stage op-amps this must be set to zero.

When specified, the optional parameters listed below override default values set via the **.MODEL** command.

- `MNAME`

  The model name, as described in the **.MODEL** command
- **LEVEL**=VAL

  1 for single-stage, 2 for two-stage op-amps. Default value is 2. The **LEVEL** parameter cannot be changed in the instantiation line, only in the **.model** card.
- **VOFF**=VAL

  Offset voltage in volts. Default value is 0 V.
- **SL**=VAL

  Slew rate in volts/second for two-stage op-amps only. Default is 1.0×10^6 V/s.

- **CIN**=VAL

  Input capacitance in farads. Default value is 0F.

- **RS**=VAL

  Output resistance in ohms. Default value is 1MΩ for single-stage and 10MΩ for two-stage op-amps.

- **VSAT**=VAL

  Symmetrical saturation voltage of ± **VSAT** in volts. Default value is 5V.

- **VSATM**=VAL

  Asymmetrical saturation voltage, with **VSATN** as lower and **VSAT** as upper saturation voltage. Default is –5V.

Note

VSAT and **VSATM** must be declared together if they are used in the instantiation line.

- **GAIN**=VAL

  Linear or dB scaling factor. Default value is 1000.

- **FC**=VAL

  Cut-off frequency in Hertz for two-stage amplifiers only. Default value is 1kHz.

- **FNDP**=VAL

  Non-dominant pole frequency in Hertz for single-stage op-amps only. Default value is 1kHz.

- **IMAX**=VAL

  Saturation current in amps for single-stage op-amps only. Default value is 100mA.

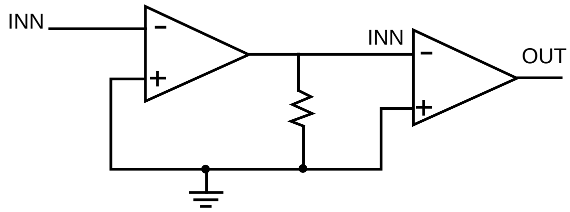
- **CMRR**=VAL

  Common mode rejection ratio, linear or in dB. Default value is zero.

Note

When using an operational amplifier macromodel, it is recommended that the default simulator accuracy (**EPS** parameter) is increased from 1mV to 1µV using the **.OPTION** command. The amplifier model calculates a current at its output and expects a voltage at its input. It is recommended, therefore that a resistor be connected between the input and output of cascaded amplifier macromodels, as shown below.

Figure 9-2. Cascaded Amplifier Macromodel

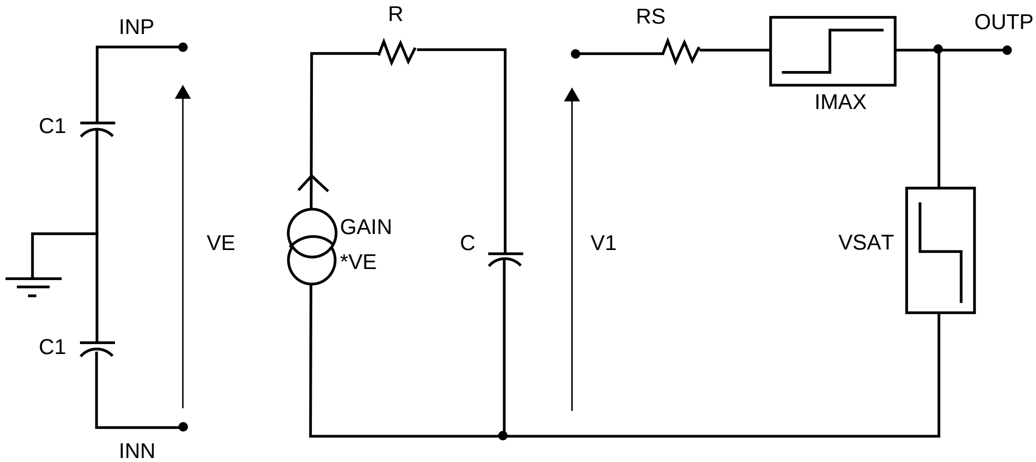


Operational Amplifier Model

```
.MODEL MNAME OPA [PAR=VAL]
```

Equivalent Circuit (Single-stage Amplifier)

Figure 9-3. Equivalent Circuit (Single-stage Amplifier)



Model Equations (Single-stage Amplifier)

R=1 kΩ which cannot be changed.

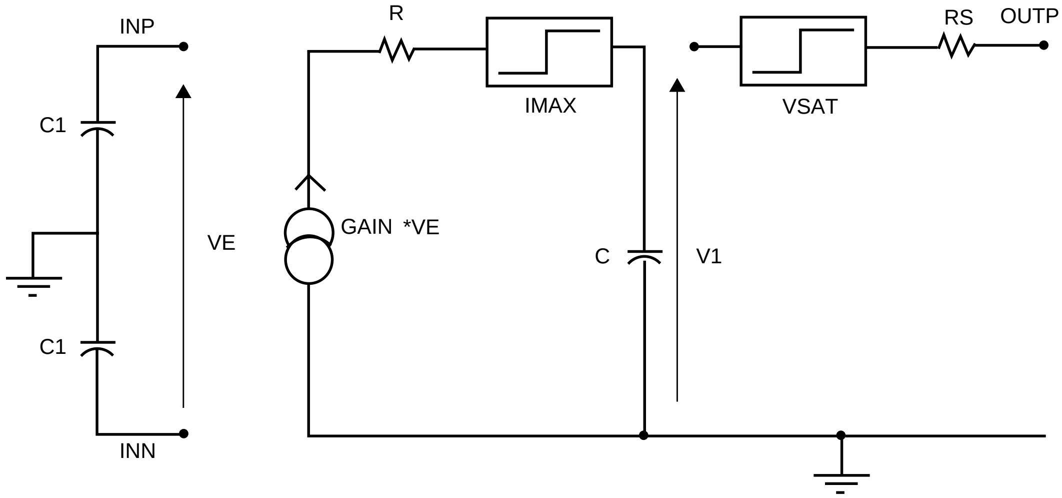
$$C = \frac{1}{2\pi \times R \times FNDP}$$

For DC analysis, ***C*** is open circuit, therefore:

$$V1 = GAIN \times VE$$

Equivalent Circuit (Two-stage Amplifier)

Figure 9-4. Equivalent Circuit (Two-stage Amplifier)



Model Equations (Two-stage Amplifier)

$\boldsymbol{R}$=1 kΩ which cannot be changed.

$$C = \frac{1}{2\pi \times R \times FC}$$

$$IMAX = SL \times C$$

For DC analysis, ***C*** is short circuit, therefore:

$$I = \frac{(GAIN \times VE)}{R}$$

If $I < IMAX$ then:

$$\mathrm{V1} = GAIN \times VE$$

else: $\mathrm{V1} = IMAX \times R$

Table 9-1. Operational Amplifier Model Parameters

Nr.	Name	Default	Units	Definition
1	LEVEL	2		Amplifier index
2	VOFF	0	V	Offset voltage
3	SL	1.0×10^6	Vs^{-1}	Slew rate (two-stage)
4	CIN	0	F	Input capacitance
5	RS	1.0×10^6	Ω	Output resistance (single-stage)
		1	Ω	Output resistance (two-stage)
6	VSAT	5	V	Symmetrical saturation voltage
7	VSATN	–5	V	Unsymmetrical saturation voltage
8	GAIN	1.0×10^3		Scaling factor
9	FC	1.0×10^3	Hz	Cut-off frequency (two-stage)
10	FNDP	1.0×10^8	Hz	Pole frequency (single-stage)
11	IMAX	0.1	A	Saturation current (single-stage)
10	CMRR	0		Common mode rejection ratio

Example

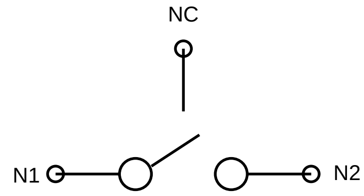
```
*OPAMP model definition
.model ampop opa level=2 voff=0 sl=50e06
+ cin=0 rs=10 vsat=5 gain=5000 fc=5000
...
*main circuit
opa1 n2 n1 n3 0 ampop
```

Specifies the operational amplifier **opa**1 of model type ampop having input nodes n2 (+ve) and n1 (–ve) with output nodes n3 (+ve) and ground (–ve). The electrical parameters of the op-amp are specified using the **.MODEL** command.

Switch

```
Sxx NC N1 N2 [MNAME] [RON [CREC]]
```

Figure 9-5. Switch Macromodel



Parameters

- `xx`

  Switch name

- `NC`

  Switch voltage controlling node

- `N1`

  Name of the node 1

- `N2`

  Name of the node 2

- `MNAME`

  Model name, as described in the **.MODEL** command

- `RON`

  "On" resistance of the switch in ohms. Default is 1kΩ. If RON≤0, then it is reset to 1kΩ.

- `CREC`

  Overlap capacitance, modeling Charge Injection. Default is zero.

Note

Switch macromodels may only be used in transient noise or DC simulations.

Example

```
s23 c n2 n6 2000 0.02e-12
```

Specifies a switch named s23 placed between nodes n2 and n6, with controlling node c, having a 2kΩ "on" resistance and 0.02pF overlap capacitance.

Switch Model

```
.MODEL MNAME NSW [PAR=VAL]         ! NMOS
.MODEL MNAME PSW [PAR=VAL]         ! PMOS
```

Model Equivalent Circuit

Figure 9-6. Closed Switch Equivalent Circuit

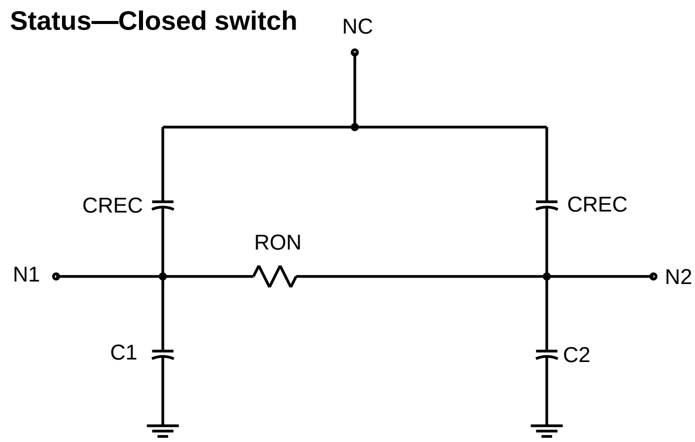


Figure 9-7. Open Switch Equivalent Circuit

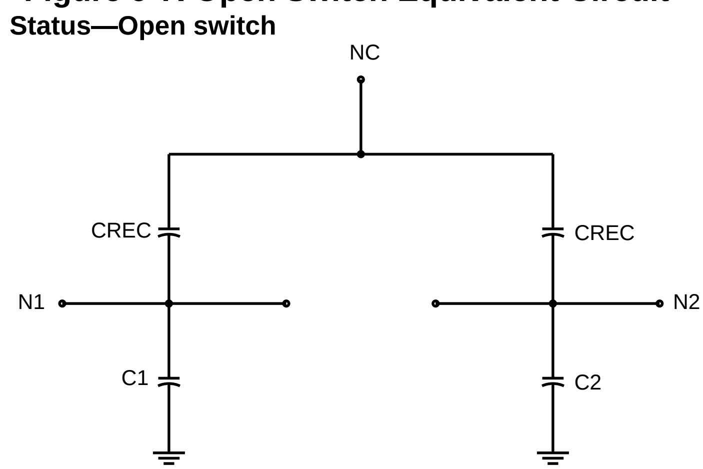


Figure 9-8. NMOS Switch

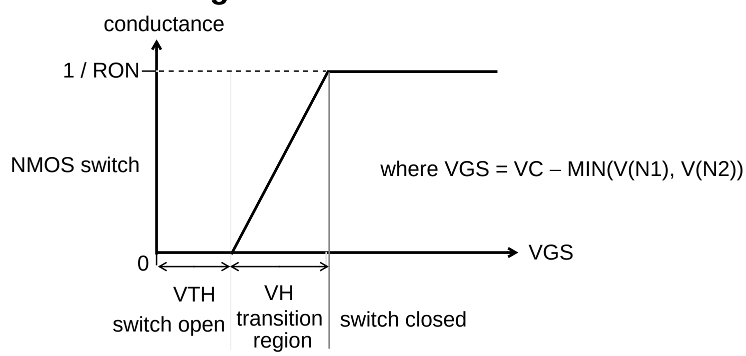


Figure 9-9. PMOS Switch

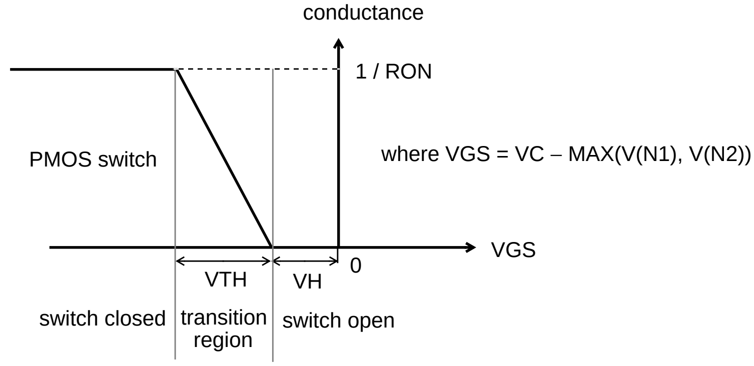


Table 9-2. Switch Model Parameters

Nr.	Name	Default	Units	Definition
1	VTH	0.47	V	Threshold voltage (for enhanced NMOS)
		–0.47	V	Threshold voltage (for enhanced PMOS)
2	VH	0.5	V	Transition voltage (for enhanced NMOS)
		–0.5	V	Transition voltage (for enhanced PMOS)
3	RON	1.0×10^{3}	Ω	"On" resistance
4	CREC	0	F	Total overlap capacitance
5	C1	10×10^{-15}	F	Switch input capacitance
6	C2	10×10^{-15}	F	Switch output capacitance

Further Explanation of Parameters

- `VTH`

  Threshold voltage. The voltage at which the "off" resistance starts changing.

- `VH`

  When the control voltage `V(NC)` reaches `VTH+VH`, the switch attains the "on" resistance.

Example

```
.model styp nsw vh=0.4 vth=0.5 ron=1k crec=50f
...
s7 ck3 5 7 styp
```

Specifies the switch s7 of model type styp (N-type) placed between the nodes 5 and 7 with controlling node ck3. The electrical parameters of the switch are specified using the **`.MODEL`** command.

Z-domain Representation

General Notes on the Use of Macromodels

The following macromodels are basic building blocks for switched capacitor filters and sampled data systems, controlled by two clock phases 1 and 2. The behavior of each SC element is characterized by an admittance matrix which is created internally during the transient or the AC analysis as a function of the parameters (capacitance `C`, period `TP` of the controlling clocks and the **`LDI`** flag). This matrix transforms the vector of all the pin voltages into the vector of all the currents entering the pins of the element. This transformation works in a (discrete) time domain as well as in a frequency domain and allows both the transient and AC analyses of the SC circuit.

Note

The controlling clock signals are represented by the parameter **`TP`**.

This modeling approach is linear, but extremely fast. Using this approach, a transient analysis can be performed within a few seconds, whereas a couple of hours would be required if the SC filter was built on a transistor circuit description level!

The following section illustrates the uses of these macromodels.

LDI Definition: LDI Transformation flag can equal 0, 1 or 2. It is only important for AC analysis. It has to be set if a switched capacitor representation of a resistor is connected to the inverting input of an op-amp, with a feedback capacitance and if both form an **`LDI`**, **`LDI`**=1 or 2 identifies the switch branch, which is connected to the inverting input of the op-amp by its equally named controlling clock-phase.

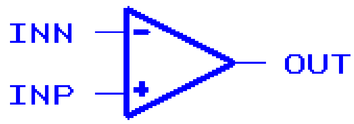
See "SC Integrators & LDI's" on page 493.

The effect is that the current of the corresponding switch branch will be delayed by a factor of z–1/2. If **`LDI`**=0, no LDI transformation will be performed.

Ideal Operational Amplifier

```
Yxx SC_IDEAL [PIN:] INP INN OUT [PARAM: M=val]
```

Figure 9-10. Ideal Operational Amplifier Macromodel



This is an ideal operational amplifier macromodel implemented for use as a basic building block in SC circuits and sampled data systems.

Model Pins

- `INP`

  Non-inverting input

- `INN`

  Inverting input

- `OUT`

  Output

- **`PARAM:`** **`M`**`=VAL`

  Device multiplier. Simulates the effect of multiple devices in parallel. In effect the current value is multiplied by **`M`**. Default is 1. **`.OPTION YMFACT`** must be selected in order for this option to work.

Note

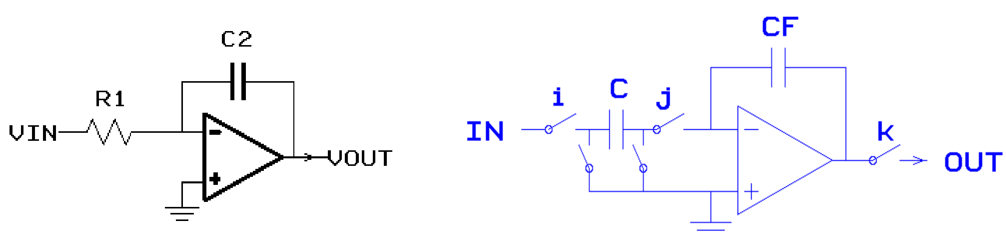
A feedback connection between output and input is recommended.

SC Integrators & LDI's

One of the most important applications of Z-domain SC elements is in the construction of SC Integrators & LDI's, the key elements for the design of switch capacitor filters and sampled data systems. In these applications they are basic building blocks, along with others such as delay elements, sample-and-hold elements, summing amplifiers etc.

But even the SC Integrators can be further decomposed into more elementary basic building blocks, namely SC resistors (`sc_i`, `sc_n`, `sc_p`, and so on), op-amps (`sc_ideal`, `opa`) and unswitched capacitors (`sc_u`). Most SC Integrators represent the same RC Integrator. They have approximately the same transfer function if the resistor R1 is substituted by an appropriate switched capacitor (`sc_i`, `sc_n`, `sc_p`, and so on). See the figure below:

Figure 9-11. SC Integrators & LDI's



Note
The transfer function of an SC Integrator depends on the sampling instants of the output signal, i.e. different clock phases ***k***, which control the switch path at the output of the integrator causing different transfer functions of this integrator.

This is now explained exactly in the formal mathematical language.

Supposing that:

> ***i*** is the phase which controls the switch branch at the input of the integrator,
> ***j*** is the phase which controls the switch branch at the inverting input of the op-amp,
>
> ***k*** is the phase which controls the switch branch at the output of the integrator, and
> H_{ijk} is the transfer function of the integrator controlled by ***i***, ***j***, ***k***.

The following relationship will be true:

$$H_{ijk}(z) = H_{ikk}(z) \times z^{\alpha}$$

where $H_{ijk}(z)$ and $H_{ikk}(z)$ are the transfer functions corresponding to the same type ***H*** of the integrator but controlled by the clock phases ***i***, ***j***, ***k*** and ***i***, ***k***, ***k*** respectively and:

$$\alpha = \begin{cases} 0 & \text{if } \boldsymbol{j}=\boldsymbol{k} \\ or & \\ -0.5 & \text{otherwise} \end{cases}$$

that is:

$$LDI = \begin{cases} 0 & \text{if } \boldsymbol{j}=\boldsymbol{k} \\ or & \\ j & \text{otherwise} \end{cases}$$

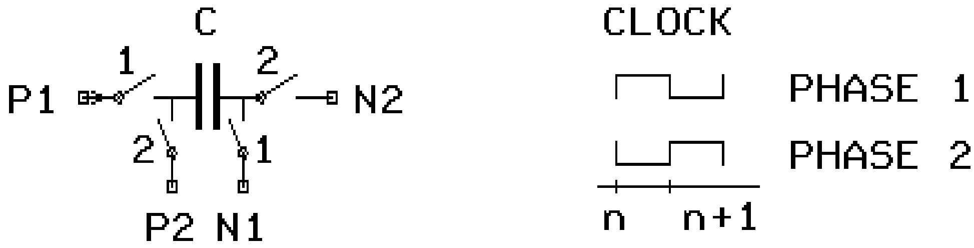
This information is taken from:
Rolf Unbehauen, Andrzej Cichocki, "*MOS Switched Capacitor and Continuous-Time Integrated Circuits and Systems.*"

For examples, please see "Applications" on page 512.

Inverting Switched Capacitor

```
Yxx SC_I [PIN:] P1 P2 N2 N1 [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 9-12. Inverting Switched Capacitor Macromodel



This is a basic building block for Z-domain modeling of SC circuits and sampled data systems. It represents an SC realization of an analog resistor, the so called inverting switched capacitor.

It has been modeled as a voltage to current four port transfer element with a characteristic transfer or admittance matrix Y(z), used for the description of the behavior of this element for both the transient and small signal AC analyses. The transfer function can be symbolically represented as shown above.

It is a Kirchhoff type element having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Only one sc_i model should be connected to an amplifier summing node (minus input). Simulation results will be incorrect if two or more sc_i macromodels are connected to the minus input node of an amplifier. In this particular case, current (charge) from output nodes of the sc_i macro is not summed together in the correct way.

Model Pins

- `P1`

  Name of the positive pin to the switch branch controlled by clock phase 1

- `P2`

  Name of the positive pin to the switch branch controlled by clock phase 2

- `N1`

  Name of the negative pin to the switch branch controlled by clock phase 1

- `N2`

  Name of the negative pin to the switch branch controlled by clock phase 2

Table 9-3. Inverting Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
1	C	1.0×10^{-12}	F	Capacitance
2	TP	1.0×10^{-6}	s	Pulse period
3	LDI	0		See the note on LDI’s in this chapter
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Note

The above macromodel parameters can be declared in a number of ways, listed below in order of priority.

1. In the instantiation line, using the **.PARAM** keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

   If this syntax is used, a model name must be declared using the **.MODEL** keyword in the instantiation line.
3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

This element performs the linear Z-domain transfer function:

$$[I_{p1}(z), I_{n1}(z), I_{p2}(z), I_{n2}(z)] = Y(z, LDI)[V_{p1}(z), V_{n1}(z), V_{p2}(z), V_{n2}(z)]$$

in the time and frequency domain for the transient and small signal AC analyses where:

I_{p1}, I_{n1}, I_{p2}, I_{n2} are the currents contributed to the pins P1, N1, P2, N2.

V_{p1}, V_{n1}, V_{p2}, V_{n2} are the voltages at the nodes P1, N1, P2, N2.

Y(z, LDI) is the characteristic transfer or admittance matrix of the model.

The Z-transfer matrix ***Y(z,0)*** is the Z-transform of the difference equations describing the time domain behavior of this element. Therefore the case of **LDI**=0 is completely adequate to the time domain behavior.

In some SC Integrators, the pins `P2`, `N1` are grounded, whereas `N2` is connected to the virtual ground of an ideal op-amp, in which case the transfer function at node `N2` becomes the following form:

$$I_{n2}(z) = \frac{C}{Tp} \times z^{-1/2} \times V_{p1}(z)$$

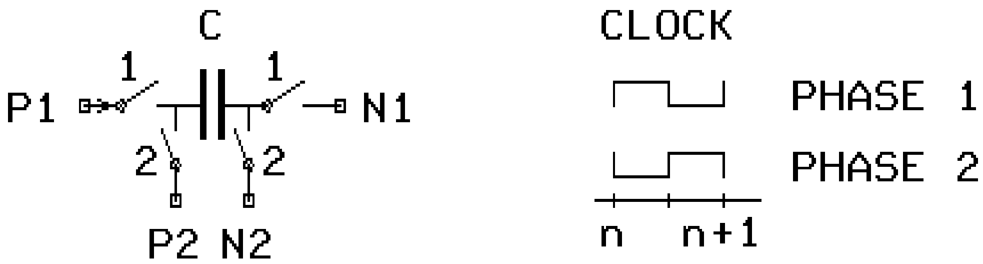
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

Non-inverting Switched Capacitor

```
Yxx SC_N [PIN:] P1 P2 N1 N2 [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 9-13. Non-inverting Switched Capacitor Macromodel



This is a basic building block for Z-domain modeling of SC circuits and sampled data systems. It represents an SC realization of an analog resistor, the so called non-inverting switched capacitor.

It has been modeled as a voltage to current four port transfer element with a characteristic transfer or admittance matrix Y(z), used for the description of the behavior of this element for both the transient and small signal AC analyses. The transfer function can be symbolically represented as shown above.

It is a Kirchhoff type element having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Model Pins

- `P1`

  Name of the positive pin to the switch branch controlled by clock phase 1
- `P2`

  Name of the positive pin to the switch branch controlled by clock phase 2
- `N1`

  Name of the negative pin to the switch branch controlled by clock phase 1
- `N2`

  Name of the negative pin to the switch branch controlled by clock phase 2

Table 9-4. Non-inverting Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
1	C	1.0×10^{-12}	F	Capacitance

Table 9-4. Non-inverting Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
2	TP	1.0×10^{6}	s	Pulse period
3	LDI	0		See the note on LDI's in this chapter
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Note

The above macromodel parameters can be declared in a number of ways, listed below in order of priority.

1. In the instantiation line, using the **.PARAM** keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

   If this syntax is used, a model name must be declared using the **.MODEL** keyword in the instantiation line.
3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

This element performs the linear Z-domain transfer function:

$$[I_{p1}(z), I_{n1}(z), I_{p2}(z), I_{n2}(z)] = Y(z, LDI)[V_{p1}(z), V_{n1}(z), V_{p2}(z), V_{n2}(z)]$$

in the time and frequency domain for the transient and small signal AC analyses where:

I_{p1}, I_{n1}, I_{p2}, I_{n2} are the currents contributed to the pins P1, N1, P2, N2.

V_{p1}, V_{n1}, V_{p2}, V_{n2} are the voltages at the nodes P1, N1, P2, N2.

Y(z, LDI) is the characteristic transfer or admittance matrix of the model.

The Z-transfer matrix ***Y*(*z*,*0*)** is the Z-transform of the difference equations describing the time domain behavior of this element. Therefore the case of **LDI**=0 is completely adequate to the time domain behavior.

In some SC Integrators the pins `P2`, `N2` are grounded whereas `N1` is connected to the virtual ground of an ideal op-amp, in which case the transfer function at node `N1` becomes the following form:

$$I_{n1}(z) = -\frac{C}{Tp} \times V_{p1}(z)$$

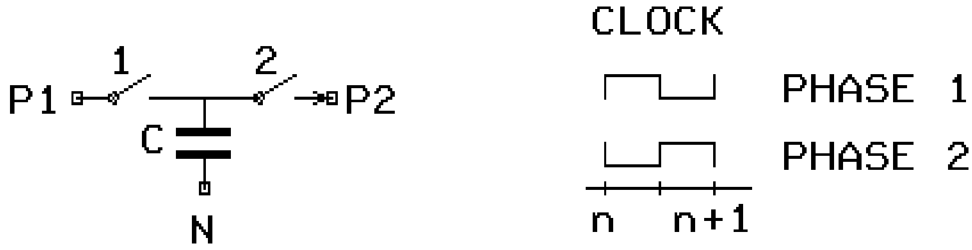
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

Parallel Switched Capacitor

```
Yxx SC_P [PIN:] P1 P2 N [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 9-14. Parallel Switched Capacitor Macromodel



This is a basic building block for Z-domain modeling of SC circuits and sampled data systems. It represents an SC realization of an analog resistor, the so called parallel switched capacitor.

It has been modeled as a voltage to current three port transfer element with a characteristic transfer or admittance matrix Y(z), used for the description of the behavior of this element for both the transient and small signal AC analyses. The transfer function can be symbolically represented as shown above.

It is a Kirchhoff type element having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Model Pins

- `P1`

  Name of the positive pin to the switch branch controlled by clock phase 1
- `P2`

  Name of the positive pin to the switch branch controlled by clock phase 2
- `N`

  Name of the negative pin at the outer side of the capacitor not connected to the switches of this element

Table 9-5. Parallel Switched Capacitor

Nr.	Name	Default	Units	Definition
1	C	1.0×10^{-12}	F	Capacitance
2	TP	1.0×10^{6}	s	Pulse period
3	LDI	0		See the note on LDI’s in this chapter

Table 9-5. Parallel Switched Capacitor

Nr.	Name	Default	Units	Definition
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Note
The above macromodel parameters can be declared in a number of ways, listed below in order of priority.

1. In the instantiation line, using the **.PARAM** keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

   If this syntax is used, a model name must be declared using the **.MODEL** keyword in the instantiation line.
3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

This element performs the linear Z-domain transfer function:

$$[I_{p1}(z), I_{p2}(z), I_n(z)] = Y(z, LDI)[V_{p1}(z), V_{p2}(z), V_n(z)]$$

in the time and frequency domain for the transient and small signal AC analyses where:

I_{p1}, I_{p2}, I_n are the currents contributed to the pins P1, P2, N.

V_{p1}, V_{p2}, V_n are the voltages at the nodes P1, P2, N.

Y(z, LDI) is the characteristic transfer or admittance matrix of the model.

The Z-transfer matrix ***Y*(*z*,*0*)** is the Z-transform of the difference equations describing the time domain behavior of this element. Therefore the case of **LDI**=0 is completely adequate to the time domain behavior.

In some SC Integrators the pin `N` is grounded whereas `P2` is connected to the virtual ground of an ideal op-amp, in which case the transfer function becomes the following form:

$$I_{p2}(z) = -\frac{C}{Tp} \times z^{-1/2} \times V_{p1}(z)$$

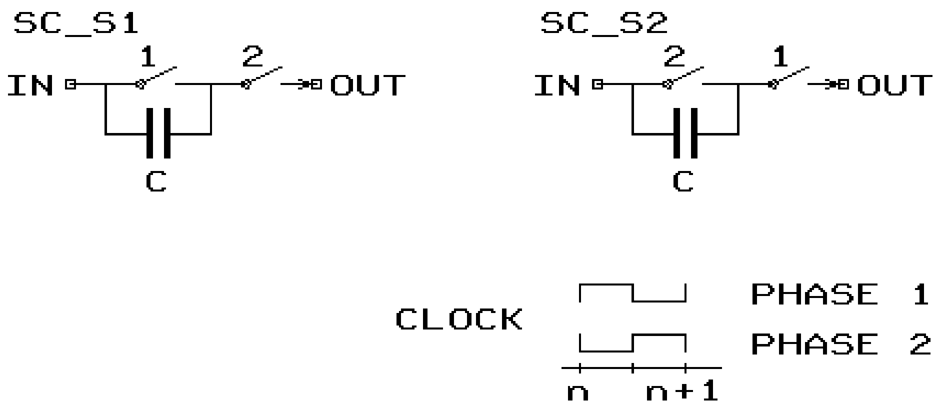
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

Serial Switched Capacitor

```
Yxx SC_S1 [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx SC_S2 [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 9-15. Serial Switched Capacitor



These are basic building blocks for the Z-domain modeling of SC circuits and sampled data systems. They represent SC realizations of an analog resistor, the so called serial switched capacitor.

They have been modeled as voltage to current two port transfer elements with a characteristic transfer or admittance matrix Y(z), used for the description of the behavior of these elements for both the transient and small signal AC analyses. The transfer function can be symbolically represented as shown above.

They are Kirchhoff type elements having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes, that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Model Pins

- `IN`

  Name of the positive pin to the switch branch controlled by clock phase 1 (or 2)

- `OUT`

  Name of the pin to the switch branch controlled by clock phase 2 (or 1)

Table 9-6. Serial Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
1	C	1.0×10^{-12}	F	Capacitance
2	TP	1.0×10^{6}	s	Pulse period

Table 9-6. Serial Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
3	LDI	0		See the note on LDI's in this chapter
4	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Note

The above macromodel parameters can be declared in a number of ways, listed below in order of priority.

1. In the instantiation line, using the **.PARAM** keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

   If this syntax is used, a model name must be declared using the **.MODEL** keyword in the instantiation line.
3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

This element performs the linear Z-domain transfer function:

$$[I_{in}(z), I_{out}(z)] = Y(z, LDI)[V_{in}(z), V_{out}(z)]$$

in the time and frequency domain for the transient and small signal AC analyses where:

I_{in}, I_{out} are the currents contributed to the pins IN, OUT.

V_{in}, V_{out} are the voltages at the nodes IN, OUT.

Y(z, LDI) is the characteristic transfer or admittance matrix of the model.

The Z-transfer matrix ***Y*(*z*,*0*)** is the Z-transform of the difference equations describing the time domain behavior of this element. Therefore the case of **LDI**=0 is completely adequate to the time domain behavior.

In some SC Integrators the pin `OUT` is connected to the virtual ground of an ideal op-amp, in which case the transfer function becomes the following form:

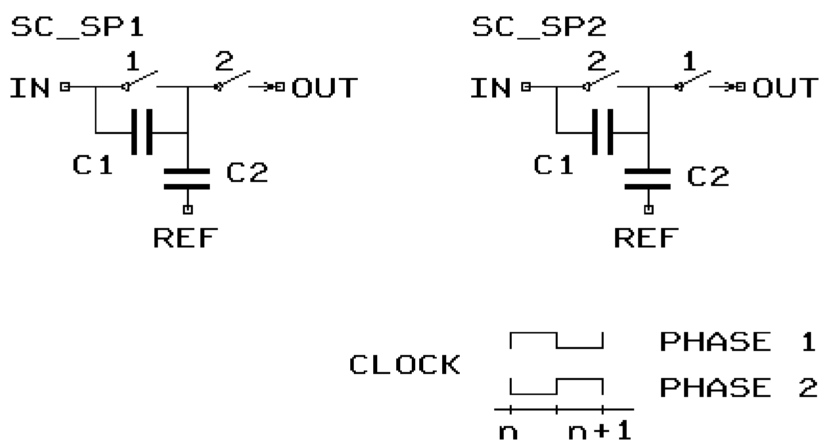
$$I_{out}(z) = -\frac{C}{Tp} \times V_{in}(z)$$

where: $z^k = e^{j\omega(k \times Tp)}, k \in Integer$

Serial-parallel Switched Capacitor

```
Yxx SC_SP1 [PIN:] IN OUT REF [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
Yxx SC_SP2 [PIN:] IN OUT REF [PARAM: PAR=VAL {PAR=VAL}][MODEL: MNAME]
```

Figure 9-16. Serial-parallel Switched Capacitor Macromodel



These are basic building blocks for the Z-domain modeling of SC circuits and sampled data systems. They represent SC realizations of an analog resistor, the so called serial-parallel switched capacitor.

They have been modeled as voltage to current three port transfer elements with a characteristic transfer or admittance matrix Y(z), used for the description of the behavior of these elements for both the transient and small signal AC analyses. The transfer function can be symbolically represented as shown above.

They are Kirchhoff type elements having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes, that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Model Pins

- `IN`

  Name of the positive pin to the switch branch controlled by clock phase 1 (or 2)

- `OUT`

  Name of the positive pin to the switch branch controlled by clock phase 2 (or 1)

- `REF`

  Name of the negative pin

Table 9-7. Serial-parallel Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
1	C1	$1.0{\times}10^{-12}$	F	Capacitance
2	C2	$1.0{\times}10^{-12}$	F	Capacitance
3	TP	$1.0{\times}10^{6}$	s	Pulse period
4	LDI	0		See the note on LDI's in this chapter
5	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Note

The above macromodel parameters can be declared in a number of ways, listed below in order of priority.

1. In the instantiation line, using the **.PARAM** keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

   If this syntax is used, a model name must be declared using the **.MODEL** keyword in the instantiation line.
3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

These elements perform the linear Z-domain transfer function:

$$[I_{in}(z), I_{out}(z), I_{ref}(z)] = Y(z, LDI)[V_{in}(z), V_{out}(z), V_{ref}(z)]$$

in the time and frequency domain for the transient and small signal AC analyses where:

I_{in}, I_{out}, I_{ref} are the currents contributed to the pins IN, OUT, REF.

V_{in}, V_{out}, V_{ref} are the voltages at the nodes IN, OUT, REF.

Y(z, LDI) is the characteristic transfer or admittance matrix of the model.

The Z-transfer matrix ***Y***(***z,0***) is the Z-transform of the difference equations describing the time domain behavior of this element. Therefore the case of **LDI**=0 is completely adequate to the time domain behavior.

In some SC Integrators the pin `N` is connected to the virtual ground of an ideal op-amp, in which case the transfer function becomes the following form:

$$I_{out}(z) = -\frac{1}{Tp}(C_1 + C_2 z^{-1/2}) \times V_{in}$$

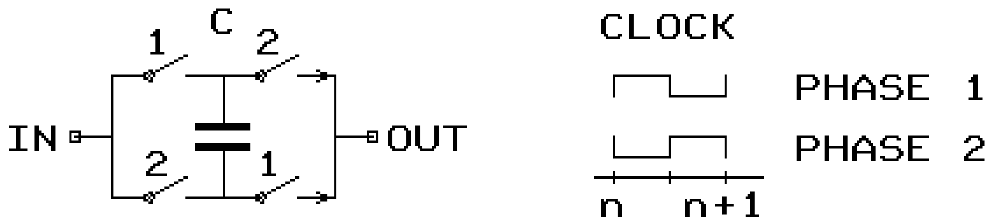
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

Bi-linear Switched Capacitor

```
Yxx SC_B [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 9-17. Bi-linear Switched Capacitor Macromodel



This is a basic building block for Z-domain modeling of SC circuits and sampled data systems. It represents an SC realization of an analog resistor, the so called bi-linear switched capacitor.

It has been modeled as a voltage to current two port transfer element with a characteristic transfer or admittance matrix Y(z), used for the description of the behavior of this element for both the transient and small signal AC analyses. The transfer function can be symbolically represented as shown in the above diagram.

It is a Kirchhoff type element having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Model Pins

- `IN`

  Name of the first pin
- `OUT`

  Name of the second pin

Table 9-8. Bi-linear Switched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
1	C	1.0×10^{-12}	F	Capacitance
2	TP	1.0×10^{6}	s	Pulse period
3	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

Note

The above macromodel parameters can be declared in a number of ways, listed below in order of priority.

1. In the instantiation line, using the `.PARAM` keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

   If this syntax is used, a model name must be declared using the `.MODEL` keyword in the instantiation line.
3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

This element performs the linear Z-domain transfer function:

$$[I_{in}(z), I_{out}(z)] = Y(z, LDI)[V_{in}(z), V_{out}(z)]$$

in the time and frequency domains for the transient and small signal AC analyses where:

I_{in}, I_{out} are the currents contributed to the pins IN, OUT.

V_{in}, V_{out} are the voltages at the nodes IN, OUT.

Y(z, LDI) is the characteristic transfer or admittance matrix of the model.

In some SC Integrators, the pin `OUT` is connected to the virtual ground of an ideal op-amp, in which case the transfer function becomes the following form:

$$I_{out}(z) = -\frac{C}{Tp} \times (1 + z^{-1}) \times V_{in}(z)$$

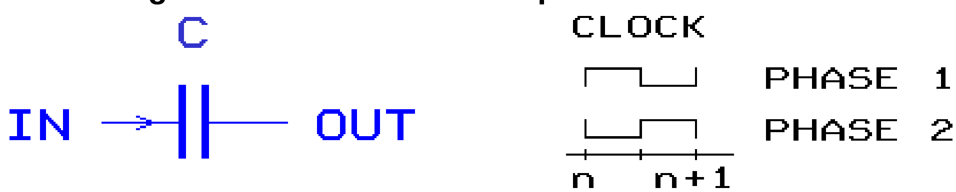
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

Unswitched Capacitor

```
Yxx SC_U [PIN:] IN OUT [PARAM: PAR=VAL {PAR=VAL}] [MODEL: MNAME]
```

Figure 9-18. Unswitched Capacitor Macromodel



This is a basic building block for Z-domain modeling of SC circuits and sampled data systems. It represents an SC realization of an unswitched capacitor.

It is a Kirchhoff type element having a time-discrete I-U-dependence with charge storage and delay effects controlled by the uniform two-phase clock.

The model assumes that all currents remain constant during each switching phase and that they only change their values at the beginning of every switching sub-interval.

Model Pins

- `IN`

  Name of the input pin
- `OUT`

  Name of the output pin

Table 9-9. Unswitched Capacitor Model Parameters

Nr.	Name	Default	Units	Definition
1	C	1.0×10^{-12}	F	Capacitance
2	TP	1.0×10^{6}	s	Pulse period
3	M[1]	1		Device multiplier

1. **.OPTION YMFACT** must be specified for **M** to work.

The above macromodel parameters can be declared in a number of ways, listed below in order of priority:

1. In the instantiation line, using the **.PARAM** keyword.
2. In a model command line. The syntax is as follows:

   ```
   .model mname modfas par=val [par=val]
   ```

If this syntax is used, a model name must be declared using the **`.MODEL`** keyword in the instantiation line.

3. If no parameters are explicitly declared, the parameter default values are used.

Model Equations

This element performs the linear Z-domain transfer function:

$$[I_{in}(z), I_{out}(z)] = Y(z)[V_{in}(z), V_{out}(z)]$$

in the time and frequency domains for the transient and small signal AC analyses where:

I_{in}, I_{out} are the currents contributed to the pins IN, OUT.

V_{in}, V_{out} are the voltages at the nodes IN, OUT.

Y(z) is the characteristic transfer or admittance matrix of the model.

The Z-transfer matrix ***Y*(*z*)** is the Z-transform of the difference equations describing the time domain behavior of this element.

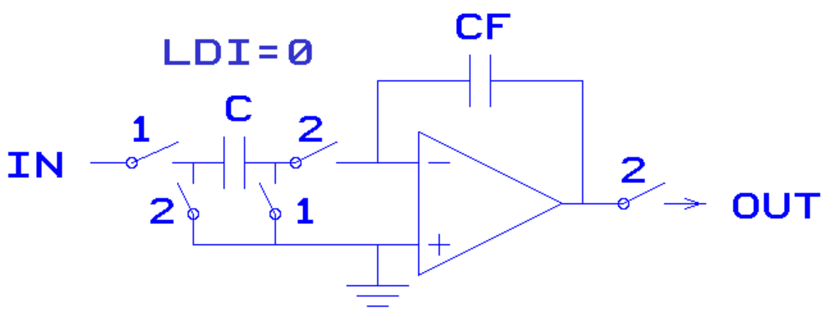
Applications

The list below gives some of the applications of Switched Capacitor Macromodels which are illustrated throughout the following pages.

- Non-inverting Integrator
- LDI Phase Control Non-inverting Integrator
- Euler Forward Integrator
- LDI Phase Control Euler Forward Integrator
- Euler Backward Integrator
- LDI Phase Control Euler Backward Integrator

Non-inverting Integrator

Figure 9-19. Non-inverting Integrator



The inverting switched capacitor is connected to the input of an op-amp with a feedback capacitor forming a non-inverting Integrator. In this configuration, both the input and output nodes of the op-amp are controlled by the same clock phase, namely clock phase 2, as shown above.

This is the *standard* use of the switched capacitor element and therefore **LDI**=0, which means that no additional delay operation has to be performed during an AC analysis.

This leads to the Z-domain transfer function:

$$vout(z) = \frac{C}{CF} \times \frac{z^{-1/2}}{1 - z^{-1}} \times vin(z)$$

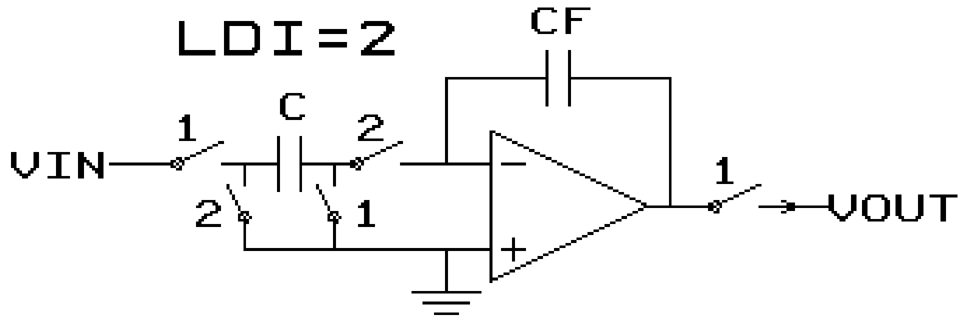
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

which describes the behavior in the time and frequency domain for the transient and small signal AC analyses.

LDI Phase Control Non-inverting Integrator

Figure 9-20. LDI Phase Control Non-inverting Integrator Macromodel



Here, the inverting switched capacitor is connected to the inverting input of an op-amp with a feedback capacitor forming a non-inverting Integrator. In this configuration the inverting input node and the output node of the op-amp are controlled by different clock phases.

This is the *non-standard* use of the switched capacitor element and therefore the **LDI** flag has to be set properly, which means that during an AC analysis, a delay operation of one half clock period has to be applied to the current through the switch branch of the SC element, which is connected to the input of the op-amp.

The **LDI** flag identifies this switch branch by the name (the number 1 or 2) of the clock phase, through which it is controlled. So **LDI**=2 in the case of the above arrangement.

This leads to the transfer function:

$$vout(z) = \frac{C}{CF} \times \frac{z^{-1}}{1 - z^{-1}} \times vin(z)$$

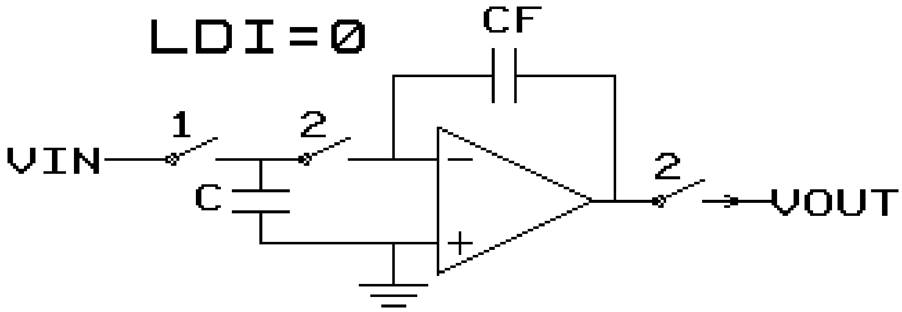
where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

which describes the behavior in the time and frequency domain for the transient and small signal AC analyses.

Euler Forward Integrator

Figure 9-21. Euler Forward Integrator Macromodel



Transfer Function

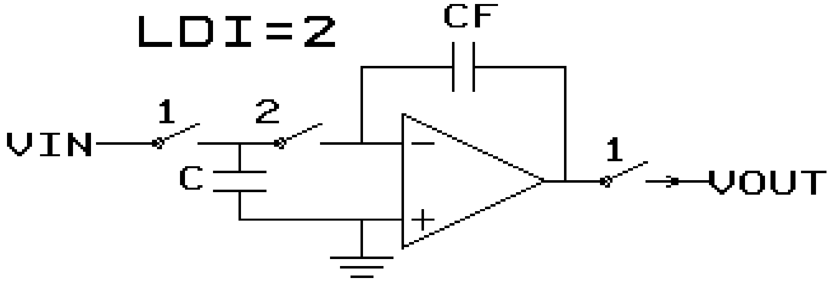
$$vout(z) = -\frac{C}{CF} \times \frac{z^{-1/2}}{1 - z^{-1}} \times vin(z)$$

where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

LDI Phase Control Euler Forward Integrator

Figure 9-22. LDI Phase Control Euler Forward Integrator Macromodel



Transfer Function

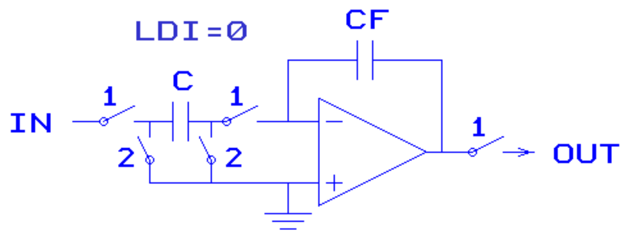
$$vout(z) = -\frac{C}{CF} \times \frac{z^{-1}}{1 - z^{-1}} \times vin(z)$$

where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

Euler Backward Integrator

Figure 9-23. Euler Backward Integrator Macromodel



Transfer Function

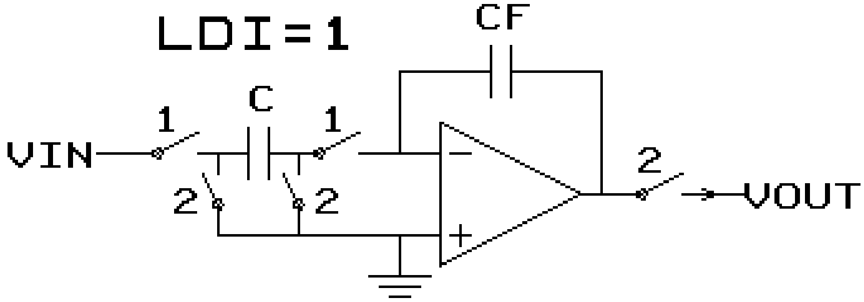
$$vout(z) = -\frac{C}{CF} \times \frac{1}{1 - z^{-1}} \times vin(z)$$

where:

$$z^k = e^{j\omega(k \times Tp)}, k \in Integer$$

LDI Phase Control Euler Backward Integrator

Figure 9-24. LDI Phase Control Euler Backward Integrator Macromodel



How to deduct the transfer function of an SC Integrator with:

- The transfer function of the SC resistor:

$$Y_{nsc}(z) = C \times vin(z) \times del_{LDI}$$

- The LDI-delay:

$$del_{LDI} = z^{-1/2}$$

- The transfer function of the unswitched capacitor in the feedback loop:

$$Y_{usc}(z) = C_F \times (1 - z^{-1})$$

- The KCL at the virtual ground node:

$$Y_{isc}(z) \times V_{in}(z) + Y_{usc}(z) \times V_{out}(z) = 0$$

We get the transfer function of the integrator as follows:

$$V_{out}(z) = -\frac{C}{CF} \times \frac{z^{-1/2}}{1 - z^{-1}} \times V_{in}(z)$$

Tutorial—SC Low Pass Filter

For a tutorial on the transient and small signal AC analysis of an SC low pass filter using the Z-domain switched capacitor models, please see "Tutorial #9—SC Low Pass Filter" on page 1369.

Chapter 10
Simulator Commands

Command Summary

In the following summary tables, commands have been organized into four groups according to their function.

- Analysis Commands
- Display Commands
- Simulation Control Commands
- Circuit Description Commands

In each of the command summary tables you can click on a command name to jump to its detailed description. The detailed command descriptions are listed in alphabetical order later in this chapter, see "Command Descriptions" on page 527.

Analysis Commands

Use these commands to instruct Eldo to perform a specific analysis.

Table 10-1. Eldo Analysis Commands

Command	Description
.AC	Activates the small signal analysis which computes the magnitude and phase of output variables as a function of frequency
.AGE	Activates the Age analysis to test the reliability of a netlist
.CHECKSOA	Causes Eldo to check for any violations of the circuit safe operating area (SOA) limits specified via the .SETSOA command
.DC	Activates a DC analysis, and is used to determine the quiescent state or operating point of the circuit
.DCHIZ	Detects high impedance values for DC analysis
.DCMISMATCH	Computes the sensitivity of node voltages and of currents through voltage sources
.DEX	Design of Experiments. Used to perform factor screening before attempting to solve subsequent statistical problems.
.DSP	Selects the waveform on which you require DSP to be applied

Table 10-1. Eldo Analysis Commands

Command	Description
.DSPMOD	Defines a set of parameters for a PSD computation using the correlogram or periodogram method. Generates a histogram of the input wave showing the wave’s magnitude probability density distribution. Provides access to the EZwave frequency measurement.
.FOUR	Selects the waveform on which the user requires FFT to be done
.LSTB	Improves the analysis of circuit stability
.MC	Performs a Monte Carlo analysis
.NOISE	Controls the noise analysis of the circuit and must be used in conjunction with an AC analysis
.NOISETRAN	Controls the transient noise analysis of a circuit and must be used in conjunction with a transient (.TRAN) analysis
.OP	Forces Eldo to determine the DC operating point of the circuit with inductors short-circuited and capacitors opened
.OPTFOUR	Used to supply the options for the FFT post-processor
.OPTIMIZE	Specifies an optimization configuration acting on all the analyses specified in the circuit netlist
.OPTNOISE	Allows more flexibility in the output of the AC noise analysis results
.PZ	For Pole-Zero analysis
.RAMP	Use when Eldo encounters difficulties finding a DC operating point with the conventional .DC command
.SENS	Causes a DC sensitivity analysis to be performed
.SENSAC	Causes an AC sensitivity analysis to be performed
.SENSPARAM	Activates a sensitivity analysis of extracts versus subcircuit parameters
.SNF	Calculates a Spot Noise Figure
.SOLVE	Solves the value of a parameter
.TF	Causes the small signal Transfer Function to be calculated by linearizing around a bias point
.TRAN	Activates a transient analysis
.WCASE	Computes worst case values for waveform data extracted using the .EXTRACT command according to a set of parameters that have LOT and DEV variations

Display Commands

Use these commands to control the messages and outputs generated from simulation.

Table 10-2. Eldo Display Commands

Command	Description
.COMCHAR	Changes the comment character on an input file
.DEFMAC	Defines a parameterized macro which may be instantiated in the circuit netlist
.DEFPLOTDIG	Enables you to plot an analog signal as a digital bus
.DEFWAVE	Defines a new waveform by relating previously defined waveforms and nodes
.EQUIV	Changes the name of a netlist node to a new display name
.EXTMOD	Performs only the .EXTRACT commands, without performing the simulation(s)
.EXTRACT	Extracts waveform information using a combination of arithmetic expressions or pre-defined functions
.FFILE	S, Y, Z Parameter output file specification
.IPROBE	Probes current between pins
.MEAS	An alternative to the .EXTRACT command
.MONITOR	Displays the steps taken by the simulator when doing an AC, TRAN or DCSWEEP simulation
.NET	Extracts the S, Y, Z or H parameters in the frequency domain for a specified circuit
.NEWPAGE	Allows the control of the saved windows file in the EZwave waveform viewer
.NOCOM	Suppresses any comment lines in the ASCII output (*.chi*) file which come after it in the netlist
.NOTRC	Suppresses the rewriting of the circuit description file in the ASCII output (*.chi*) file
.OBJECTIVE	Based on the `.EXTRACT` construct, and specifically dedicated to optimization
.OP_DISPLAY	DC operating point display. Enables the printing of alternative voltage threshold (VT) for MOS devices. Used in conjunction with a `.OP` command.
.PLOT	Specifies which simulation results have to be kept by the simulator for graphical viewing and post-processing
.PLOTBUS	Plots all the bits of a bus

Table 10-2. Eldo Display Commands

Command	Description
.PRINT	Defines the contents of a tabular listing of any number of output variables. Results are written to the *.chi* file in ASCII.
.PRINTBUS	Prints all the bits of a bus
.PRINTFILE	Defines the contents of a tabular listing of any number of output variables, and dumps them to the specified file in ASCII format or as a .DATA statement
.PROBE	Defines the contents of a tabular listing of any number of output variables. Results are written to *.wdb* binary output files for post-processing.
.PROBEBUS	Plots all the bits of a bus
.WIDTH	Sets the paper width of the output print device

Simulation Control Commands

Use these commands to modify general options for simulation.

Table 10-3. Eldo Simulation Control Commands

Command	Description
.CALL_TCL	Performs a single call to a Tcl function
.CHECKBUS	Checks for a specified value of the bus at a specified time
.CONSO	Computes and displays the average current flowing through the specified voltage source(s) during the simulation period
.CORREL	Specifies correlations between parameters during a Monte Carlo analysis
.DATA	Performs a parameter sweep
.DISCARD	Specifies whether one or more subcircuits, instances or devices should be ignored during an Eldo simulation
.DISFLAT	Disables the flat netlist mode
.DISTRIB	Specifies user-defined distributions for the device tolerances DEV and LOT used in Monte Carlo analysis
.FFILE	Specifies S, Y, Z parameter output file
.FILTER	Provides filtering capability for simulations, extractions, or to create subsets of .DATA statements
.FORCE	Forces one or more nodes to specified voltage(s) with respect to ground for the initial transient solution
.FUNC	Defines functions used in expressions

Table 10-3. Eldo Simulation Control Commands

Command	Description
.GUESS	Aids the calculation of the DC operating point by setting voltage values at selected nodes for the first iteration of a DC operating point calculation
.IC	Fixes node voltages for the duration of a DC analysis
.INIT	Initializes digital circuits
.LOAD	Takes a set of previously saved voltages and inserts these as .NODESET, .GUESS or .IC commands
.LOTGROUP	Allows different entities to share the same distribution
.MCMOD	Specifies the amount of variation on a given model parameter using the LOT and/or DEV parameters. Used in combination with Monte Carlo and Worst Case analyses.
.MODDUP	Tells Eldo that for each device_name a private .MODEL command be created. Useful in connection with SimPilot/Aspire only
.MPRUN	Runs multiple simulations on one multi-processor machine, or on many machines
.NODESET	Aids the calculation of the DC operating point by initializing selected nodes during the first DC operating point calculation
.NOISE_CORREL	Specifies correlations between independent noise sources
.NWBLOCK	Allows you to control the way Eldo will partition the netlist into several Newton Blocks
.OPTION	Allows you to modify Eldo execution behavior by allowing the setting of parameter values other than the default ones
.OPTPWL	Allows some precision parameters to be reset during transient simulation for different time windows
.OPTWIND	Allows some precision parameters to be reset during transient simulation for different time windows
.PARAM	Assigns values to parameter variables used in model and device instantiation statements
.PARAMDEX	Assigns values to parameters (*factors*) used in design of experiments
.PARAMOPT	Assigns values to parameter variables used in optimization
.RESTART	Restarts a simulation run with information previously saved using either the .SAVE or .TSAVE command
.SAVE	Writes information at specific times during simulation to a file
.SETBUS	Creates a bus with a number of bits
.SETSOA	Specifies the safe operating area limits for device parameters, model parameters and Eldo expressions

Table 10-3. Eldo Simulation Control Commands

Command	Description
.START_TIME	Forces simulation start time
.STEP	Performs several simulations while sweeping one circuit parameter or several circuit parameters simultaneously
.SUBDUP	Informs Eldo of the duplicate parameters and models which are local to the subcircuit. Also allows access to parameters in the hierarchical form for SimPilot. Useful when Eldo is running in conjunction with SimPilot.
.TABLE	Defines tables of run time values to be used in AC, DC or TRANsient analyses
.TCL_WAVE	Defines a new waveform using a Tcl function
.TEMP	Executes several successive simulations at various temperatures
.TSAVE	Saves the state of the simulation at a specified time point
.USE	Takes a set of voltages previously saved using the .SAVE <fname> DC command, and inserts these as .NODESET, .GUESS or .IC values at the point of the circuit where this command is present
.USE_TCL	Loads a Tcl file to the Eldo Tcl interpreter

Circuit Description Commands

Use these commands to describe the attributes of the circuit.

Table 10-4. Eldo Circuit Description Commands

Command	Description
.A2D	Analog-to-Digital Converter. Establishes the interface connection between Eldo and digital solvers
.ADDLIB	Searches a directory for files with file extensions *.mod* and *.ckt* (or *.sub*) and includes in the circuit description (*.cir*) file the contents of those files whose names correspond to model and subcircuit definitions (respectively) referenced and not defined in the *.cir* file
.AGE_LIB	Defines a set of functions which describe a reliability process for MOS models
.AGEMODEL	Allows you to specify the model parameters related to reliability calculations
.ALTER	Re-runs Eldo with a modified netlist
.BIND	Changes one SPICE description to another equivalent description without editing the description itself

Table 10-4. Eldo Circuit Description Commands

Command	Description
.BINDSCOPE	Specifies the hierarchical path to the instance(s) that will be replaced using the .BIND command
.CHRENT	Generates a Piece Wise Linear source using straight lines between specified points
.CHRSIM	Allows you to use the output of previous simulations as input to the current simulation
.CONNECT	Connects two nodes without modifying the circuit description file
.D2A	Digital-to-Analog Converter. Establishes the interface connection between digital solvers and Eldo
.DEFAULT	Resets the default values for elements, device initial conditions and model parameters
.DEFMOD	Maps model names in a netlist to model names specified in .MODEL cards
.DEL	Removes a library name from the nominal description
.DSPF_INCLUDE	Adds the parasitic elements described inside a specified Detailed Standard Parasitic Format (DSPF) file
.END	Marks the end of a simulator input description
.ENDL	Marks the end of a library variant description
.ENDS	Marks the end of an Eldo subcircuit description. See also .SUBCKT
.GLOBAL	Declares global node(s), making them known throughout a circuit without having to declare them in each subcircuit
.HDL	Compile and load Verilog-A modules
.HIER	Changes the hierarchy separator
.IGNORE_DSPF_ON_NODE	Forces Eldo to ignore parasitic elements on a specified node when a DSPF file has been included using the .DSPF_INCLUDE command
.INCLUDE	Inserts the contents of a file into a circuit description file
.LIB	Inserts model or subcircuit definitions into an input netlist from a library file
.LOOP	Inserts a feedback loop between the input and output nodes of an op-amp during transient analysis
.MALIAS	Maps model (or subcircuit) names in a netlist to model (or subcircuit) names specified in .MODEL or .SUBCKT cards

Table 10-4. Eldo Circuit Description Commands

Command	Description
.MAP_DSPF_NODE_NAME	Maps node references to a new DSPF node name
.MODEL	Groups sets of pre-defined parameters which may be used by one or more devices
.MODLOGIC	For the definition of digital gate models. NOTE: This is now obsolete syntax and should no longer be used. It has been retained for compatibility reasons only.
.MSELECT	Allows you to select models automatically for MOS devices
.PART	Instructs Eldo to use the specified algorithm in place of the regular transient algorithm, for a certain selection of instances
.PROTECT	Marks the start of a section of a netlist which will not be copied into the output file. Used with .UNPROTECT.
.SCALE	Scales device and model parameters of active devices automatically
.SELECT_DSPF_ON_NODE	Forces Eldo to select parasitic elements on a specified node when a DSPF file has been included using the .DSPF_INCLUDE command
.SETKEY	Defines a key (password), which is associated to a specific model or to all the models of a library
.SIGBUS	Sets signals on a bus that has been previously defined via the .SETBUS command
.SINUS	Specifies a sine wave
.SUBCKT	Marks the start of an Eldo subcircuit definition. See also .ENDS.
.TITLE	May be used to define the title of the binary output file
.TOPCELL	Selects the TOP cell subcircuit
.TVINCLUDE	Allows you to define a bus, specify inputs and check output values using a test vector file
.UNPROTECT	Marks the end of a section of a netlist which will not be copied into the output file. Used with .PROTECT.
.USEKEY	Provides the encryption key (password) to be used to allow the UDRM API to retrieve an encrypted model parameter’s value
.USE_VERILOGA	Compile and load Verilog-A modules
.VEC	Loads a test vector file
.VERILOG	Compile and load Verilog-A modules

Command Line Help

Usage

eldo -help [**commands** | **devices** | **sources** | **manual**]

Description

Online help accessible from the command line.

Arguments

Entering the command without any option displays the list of available topics.

- **commands**

  Opens a link document in Acrobat Reader, which will then allow you to select the command you require information on.

- **devices**

  Opens a link document in Acrobat Reader, which will then allow you to select the device model you require information on.

- **sources**

  Opens a link document in Acrobat Reader, which will then allow you to select the source or macromodel you require information on.

- **manual**

  Opens the full *Eldo User's Manual*, this manual, as a PDF file.

Command Descriptions

The remaining pages in this chapter describe, in alphabetical order, the Eldo simulator commands.

.A2D

Analog-to-Digital Converter

```
.A2D [SIM=simulator] eldo_node_name
+ [digital_node_name] [MOD=model_name] [parameters_list]
```

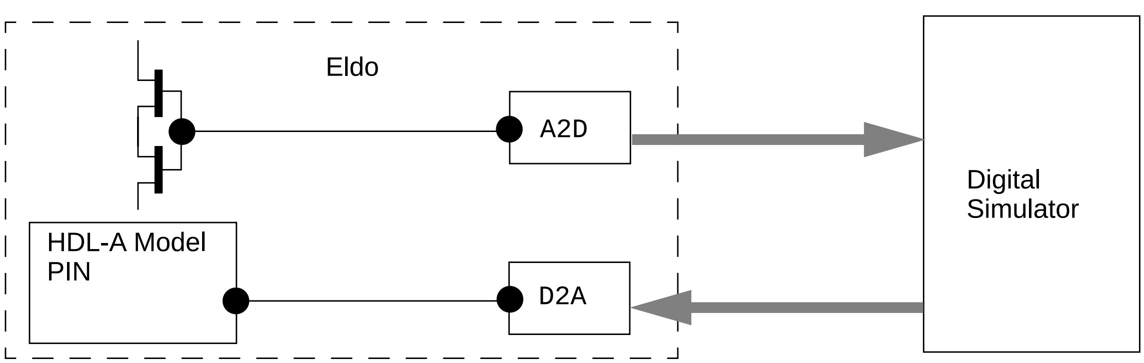
The **`.A2D`** and **`.D2A`** statements establish the interface connection between Eldo and digital solvers.

For Digital-to-Analog, please refer to the ".D2A" on page 577.

Currently, the supported digital solvers are HDL-A and Verilog.

- For Eldo/HDL-A, the converters may be used instead of an equivalent user's defined HDL-A model.
- For Eldo/Verilog, communication is done via the Cadence tool Verimix.

Note
For Eldo/Verilog communication done via the Cadence tool Verimix, the A2D and D2A syntax follows the grammar specified in the Cadence manual.



These converters transfer 'analog' Eldo information to some predefined 'digital' types. The following types are implemented: `BIT`, `X01Z`, `STD_LOGIC` and `REAL`.

`BIT (SLOPE)`	Two logical values are possible: '0' and '1'.
`X01`	Three logical values are possible: 'X', '0' and '1'.
`X01Z (RC)`	Four logical values are possible: 'X', '0', '1' and 'Z'.
`STD_VSRC`	Same values as type `X01Z`. Eldo treats this type as the type `X01Z` for the `A2D`, but as the type `STD_LOGIC` for the `D2A`.

MVL4 — Same values as type X01Z. Eldo treats this type as the type X01Z for the A2D, but as the type BIT (Voltage source) for the D2A.

STD_LOGIC (MVL9) — Nine available logical values: ‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’ ‘H’ and ‘-’. This type corresponds to the VHDL std_logic type defined in the STD_LOGIC_1164 multi-value logic system package in the IEEE library.

REAL — Real or float values.

Global Parameters

The following table shows a global view of the parameters of the **.A2D**:

Table 10-5. .A2D - Global Parameters

Name	Description	Default	Units
Common for all MODEs			
CIN	Capacitance of the digital gate input seen by Eldo	0	F
MODE= BIT | STD_VSRC			
VTH	Voltage threshold value for single threshold mode	2.5	V
VTHREL	Defines the ratio between VLOREF and VHIREF to set VTH. The value specified must be between 0.0 and 1.0. Voltage threshold value is calculated using the equation[2].	0.5	-
VLOREF	Node voltage used in conjunction with VTHREL to set VTH. Voltage threshold value is calculated using the equation[2].	-	-
VHIREF	Node voltage used in conjunction with VTHREL to set VTH. Voltage threshold value is calculated using the equation[2].	-	-
MODE= X01 | X01Z | MVL4 | STD_VSRC (FAST_STD_LOGIC)			
VTH1[1]	Lower voltage threshold value for two threshold mode, corresponding to the logical ‘0’ state	1.5	V
VTH2[a]	Higher voltage threshold value for two threshold mode, corresponding to the logical ‘1’ state	3.5	V
VTH1REL[2]	Defines the ratio between VLOREF and VHIREF to set VTH1. The value specified must be between 0.0 and 1.0. Voltage threshold value is calculated using the equation[2].	0.25	-
VTH2REL[2]	Defines the ratio between VLOREF and VHIREF to set VTH2. The value specified must be between 0.0 and 1.0. Voltage threshold value is calculated using the equation	0.75	-
VLOREF	Node voltage used in conjunction with VTH1REL and VTH2REL to set the voltage thresholds. Voltage threshold value is calculated using the equation[2].	-	-

Table 10-5. .A2D - Global Parameters

Name	Description	Default	Units
VHIREF	Node voltage used in conjunction with VTH1REL and VTH2REL to set the voltage thresholds. Voltage threshold value is calculated using the equation[2].	-	-
TX	The ‘X’ state will be sent to the DIGITAL part only if the convertor remains in the ‘X’ state for more than TX seconds	0.0	s
RZ	Effective resistance when input is ‘Z’	∞	Ω
MODE= STD_LOGIC			
STR	Defines the strength of the logic of the digital node. STRONG strength generates ‘X’, ‘0’, ‘1’, and WEAK strength generates ‘W’, ‘H’, ‘L’	STRONG	
VTH	Voltage threshold value for single threshold mode	2.5	V
VTHREL	Defines the ratio between VLOREF and VHIREF to set VTH. The value specified must be between 0.0 and 1.0. Voltage threshold value is calculated using the equation[2].	0.5	-
VTH1	Lower voltage threshold value for two threshold mode, corresponding to the logical ‘L’ state if STR=WEAK, or ‘0’ if STR=STRONG	1.5	V
VTH2	Higher voltage threshold value for two threshold mode, corresponding to the logical ‘H’ state if STR=WEAK, or ‘1’ if STR=STRONG	3.5	V
VTH1REL	Defines the ratio between VLOREF and VHIREF to set VTH1. The value specified must be between 0.0 and 1.0. Voltage threshold value is calculated using the equation[2].	0.25	-
VTH2REL	Defines the ratio between VLOREF and VHIREF to set VTH2. The value specified must be between 0.0 and 1.0. Voltage threshold value is calculated using the equation[2].	0.75	-
VLOREF	Node voltage used in conjunction with VTH1REL and VTH2REL to set the voltage thresholds. Voltage threshold value is calculated using the equation[2].	-	-
VHIREF	Node voltage used in conjunction with VTH1REL and VTH2REL to set the voltage thresholds. Voltage threshold value is calculated using the equation[2].	-	-
TX	The ‘X’ state will be sent to the DIGITAL part only if the boundary element remains in the ‘X’ state for more than TX seconds	0.0	s
R	Resistive part of the impedance of the digital gate	∞	Ω
C	Capacitive part of the impedance of the digital gate	0.0	F

Table 10-5. .A2D - Global Parameters

Name	Description	Default	Units
RZ	Effective resistance when input is ‘Z’	∞	Ω
MODE= REAL			
EPS	Defines the real value threshold	$5x10^{-3}$	V

1. **vth1** has to be specified lower than or equal to **vth2**. If not, the simulator will automatically invert **vth1** and **vth2** values so that **vth1** <= **vth2**.
2. Threshold voltage = v(vloref) + (v(vhiref) - v(vloref)) × vth*n*rel, where vth*n*rel can be vthrel, vth1rel, or vth2rel.

Parameters

- **SIM**=simulator

  The parameter **SIM** can take the value of the digital simulator's name: HDLA or VERILOG. The parameter **SIM** is mandatory only for HDL-A.

- eldo_node_name

  Name of the node in the analog netlist.

- digital_node_name

  Name of the node in the digital description. This parameter is optional. For HDL-A, the syntax is Yxx:position where position is the index of the port.

- **MOD**=model_name

  Name of the model used for the convertor. If model_name is specified, there must be a corresponding **.MODEL** command:

  ```
  .MODEL model_name A2D|ATOD parameters_list
  ```

- parameters_list

  List of available parameters as described below.

Note

For eldo_node_name connected to a PORT of an HDL-A model, parameters_list is ignored. Default values can depend on simulators (HDL-A, Verilog, and so on).

Parameters for A2D converters (.A2D)

For **.A2D** the default MODE is X01 for all simulators when two thresholds are specified. If only one threshold is specified, the default MODE is BIT.

Parameters Common for all Modes

- **CIN**=value

  Capacitance of the digital gate input seen by Eldo (in Farads). Default value is 0.0.

- **TX**=value

  If **TX** is positive, then when the convertor mode is *not* of type BIT (that is, when the convertor can generate logical ‘X’ states) the ‘X’ state will be sent to the DIGITAL part only if the convertor remains in the ‘X’ state for more than **TX** seconds. Default value is 0.0.

Parameters for MODE=BIT | STD_VSRC (FAST_STD_LOGIC)

- **VTH**=value

  Voltage threshold value for single threshold mode. The value can be specified using the parameters **VLOREF**, **VHIREF**, and **VTHREL**. Default value is 2.5V.

- **VTHREL**=value

  Defines the ratio between the value of **VLOREF** and the value of **VHIREF** to set **VTH**. The value specified must be between 0.0 and 1.0. Default value is 0.5. If defined the value on **VTH** will be ignored. **VTH** is calculated using the equation:

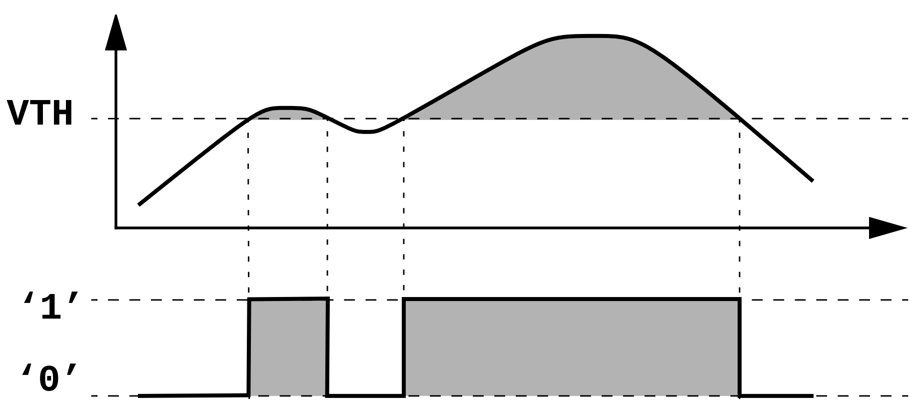
  VTH= v(**VLOREF**) + (v(**VHIREF**) - v(**VLOREF**)) × **VTHREL**

- **VHIREF**=node_name

  Node voltage used in conjunction with **VTHREL** to set **VTH**.

- **VLOREF**=node_name

  Node voltage used in conjunction with **VTHREL** to set **VTH**.



Parameters for MODE=X01 | X01Z | MVL4 | STD_VSRC (FAST_STD_LOGIC)

- **VTH1|VOLTLOW**=value

  Lower voltage threshold value for two threshold mode, corresponding to the logical ‘0’ state. The value can be specified using the parameters **VLOREF**, **VHIREF** and **VTH1REL**. Default value is 1.5V.

- **VTH2|VOLTHIGH**=value

  Higher voltage threshold value for two threshold mode, corresponding to the logical '1' state. The value can be specified using the parameters **VLOREF**, **VHIREF** and **VTH1REL**. Default value is 3.5V.

- **VTH1REL**=value

  Defines the ratio between **VLOREF** and **VHIREF** to set **VTH1**. The value specified must be between 0.0 and 1.0. Default value is 0.25. **VTH1** is calculated using the equation:

  VTH1= v(**VLOREF**) + (v(**VHIREF**) - v(**VLOREF**)) × **VTH1REL**

- **VTH2REL**=value

  Defines the ratio between **VLOREF** and **VHIREF** to set **VTH2**. The value specified must be between 0.0 and 1.0. Default value is 0.75. **VTH2** is calculated using the equation:

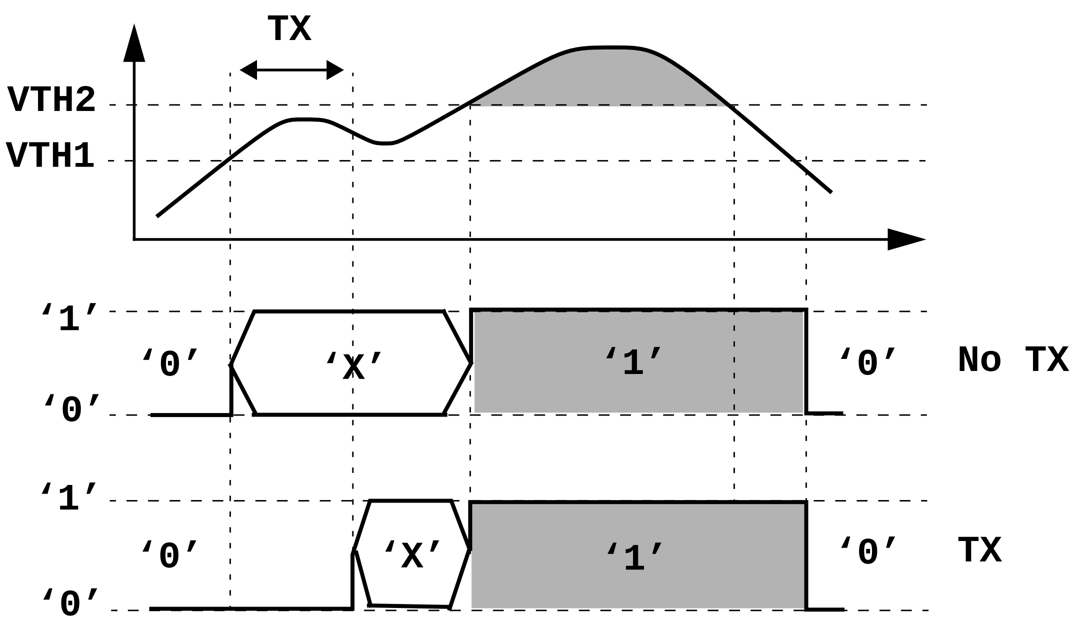
  VTH2= v(**VLOREF**) + (v(**VHIREF**) - v(**VLOREF**)) × **VTH2REL**

- **VHIREF**=node_name

  Node voltage used in conjunction with **VTH1REL** and **VTH2REL** to set **VTH1** and **VTH2**.

- **VLOREF**=node_name

  Node voltage used in conjunction with **VTH1REL** and **VTH2REL** to set **VTH1** and **VTH2**.



Note

An analog value falling between **VTH1** and **VTH2** gives a logical 'X' state. For **MODE**=MVL4, there is no analog values that can generate a logical 'Z' state.

Parameters for MODE=STD_LOGIC

- **STR**=**WEAK**|**STRONG**

  Defines the strength of the logic of the digital node. **STRONG** strength generates 'X', '0', '1', and **WEAK** strength generates 'W', 'H', 'L'. Default is **STRONG**.

- **VTH**=value

  Voltage threshold value for single threshold mode. The value can be specified using the parameters **VLOREF**, **VHIREF**, and **VTHREL**. Default value is 2.5V.

- **VTHREL**=value

  Defines the ratio between the value of **VLOREF** and the value of **VHIREF** to set **VTH**. The value specified must be between 0.0 and 1.0. Default value is 0.5. If defined the value on **VTH** will be ignored. **VTH** is calculated using the equation:

  $$VTH = v(VLOREF) + (v(VHIREF) - v(VLOREF)) \times VTHREL$$

- **VTH1**|**VOLTLOW**=value

  Lower voltage threshold value for two threshold mode, corresponding to the logical 'L' state if **STR**=**WEAK**, or '0' if **STR**=**STRONG**. The value can be specified using the parameters **VLOREF**, **VHIREF** and **VTH1REL**. Default value is 1.5V.

- **VTH2**|**VOLTHIGH**=value

  Higher voltage threshold value for two threshold mode, corresponding to the logical 'H' state if **STR**=**WEAK**, or '1' if **STR**=**STRONG**. The value can be specified using the parameters **VLOREF**, **VHIREF** and **VTH1REL**. Default value is 3.5V.

- **VTH1REL**=value

  Defines the ratio between **VLOREF** and **VHIREF** to set **VTH1**. The value specified must be between 0.0 and 1.0. Default value is 0.25. **VTH1** is calculated using the equation:

  $$VTH1 = v(VLOREF) + (v(VHIREF) - v(VLOREF)) \times VTH1REL$$

- **VTH2REL**=value

  Defines the ratio between **VLOREF** and **VHIREF** to set **VTH2**. The value specified must be between 0.0 and 1.0. Default value is 0.75. **VTH2** is calculated using the equation:

  $$VTH2 = v(VLOREF) + (v(VHIREF) - v(VLOREF)) \times VTH2REL$$

- **VHIREF**=node_name

  Node voltage used in conjunction with **VTH1REL** and **VTH2REL** to set **VTH1** and **VTH2**.

- **VLOREF**=node_name

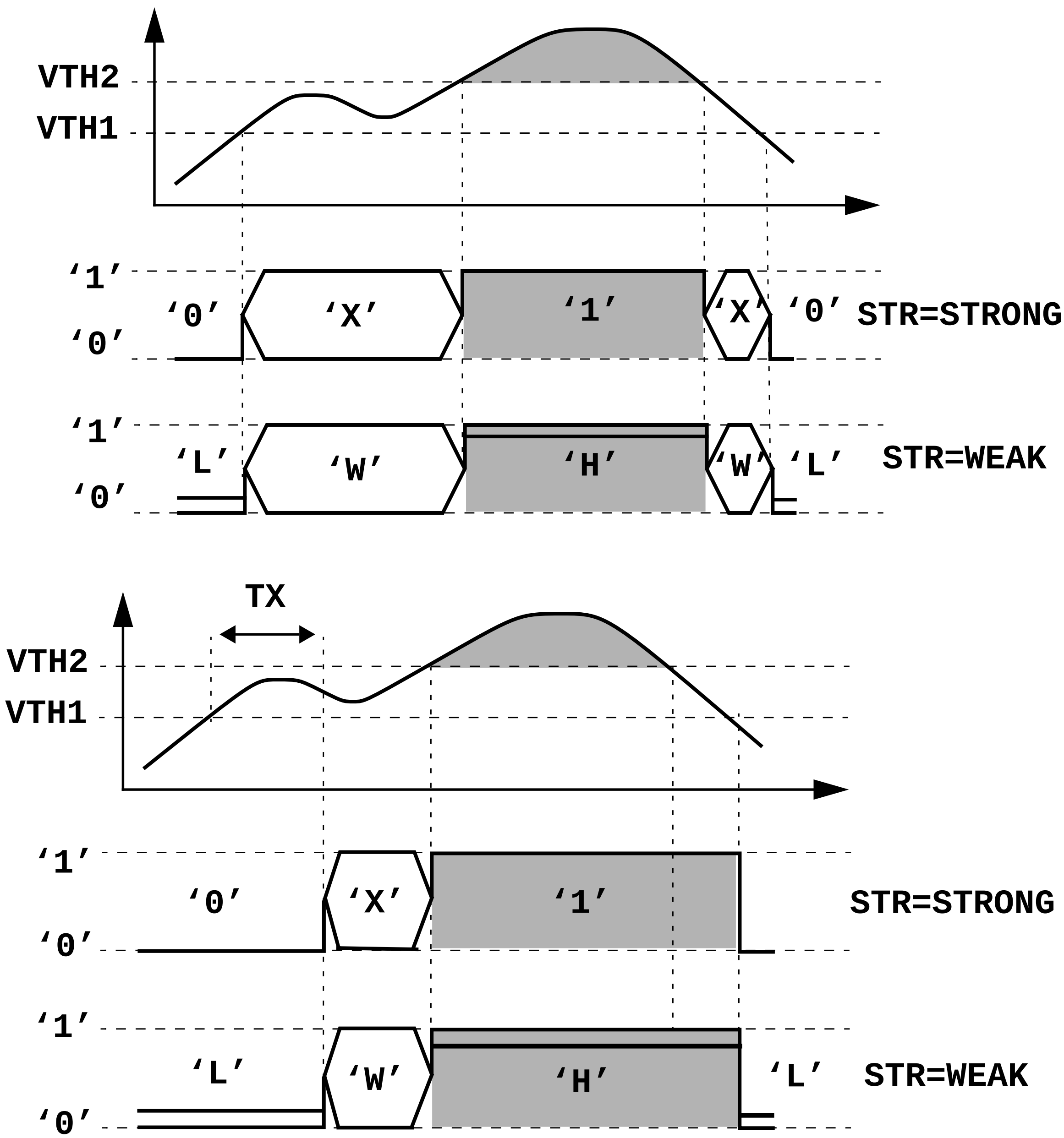
  Node voltage used in conjunction with **VTH1REL** and **VTH2REL** to set **VTH1** and **VTH2**.

- **R**=value

  Resistive part of the impedance of the digital gate. Default value is infinity.

- `C=value`

  Capacitive part of the impedance (in Farads) of the digital gate. Default value is 0.0.



Note

An analog value falling between **VTH1** and **VTH2** generates a logical 'W' state if **STR=WEAK**, and a logical 'X' state if **STR=STRONG**.

Z state detection on A2D nodes

To enable Eldo to detect an analog 'Z' state on A2D nodes, specify the **.OPTION ZDETECT** command within the netlist. This detection will *only* be performed on the A2D nodes for which an **RZ** value has been specified (either in the A2D card or via the **.model A2D** command). If the *global* value for **RZ** is set using the **.OPTION RZ**=val command then this will enable Eldo to detect a ‘Z’ state on *all* A2D nodes. A ‘Z’ state is detected when the equivalent impedance of the A2D node exceeds the specified **RZ** value.

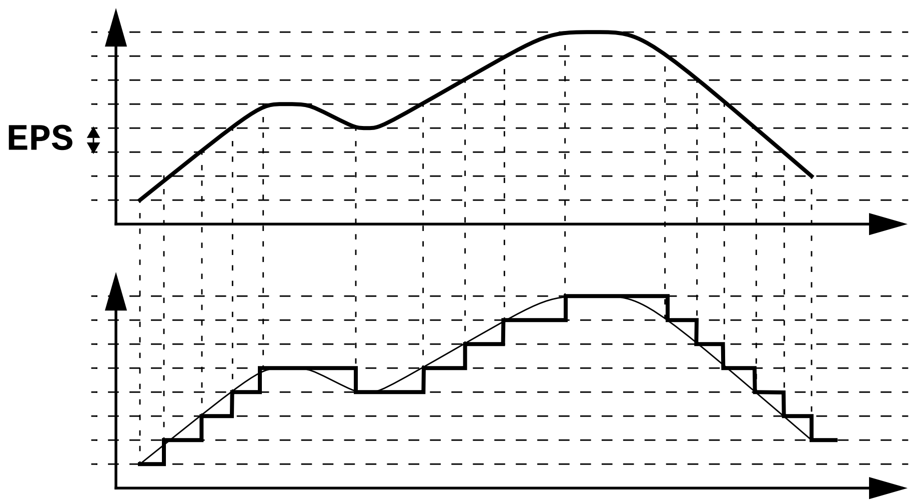
Notes on usage

- Using the **.OPTION ZDETECT** command to detect the ‘Z’ state will result in extra CPU usage, which is typically an added few percent of the total simulation time.
- The detection of the ‘Z’ state is very sensitive to the choice of the time step around the time at which the ‘Z’ state should be detected. In some cases there may be a delay between, when the ‘Z’ state is *detected*, and when the ‘Z’ state *occurs*.

Parameters for MODE=REAL

- **EPS**=value

  Defines the real value threshold. When the real value coming from the digital simulator differs from the previous value by more than **EPS**, Eldo will take this into account. When the difference is less than **EPS**, the digital output is considered to be constant.



Option DEFA2D

It is normally necessary to specify an explicit A2D between ANALOG nodes and HDL-A ports; however, with the following statement specified:

```
.OPTION DEFA2D=model_name
```

Eldo will automatically insert an implicit A2D convertor when needed. The following must be defined in the netlist:

```
.MODEL model_name A2D parameters
```

Furthermore, it is often necessary to specify **MODE**=BIT in the **.MODEL** model_name command. This is because the default for **MODE** is X01, and very often HDL-A PORTS are of type **BIT**, hence a resulting error mismatch in convertor type would be printed if **MODE** is not correctly specified. Example:

```
.MODEL model_name A2D mode = BIT
.OPTION DEFA2D = model_name
```

Additionally, these default models must be found in the netlist itself, and cannot be specified via **.LIB** or **.ADDLIB** specifications.

Note

With the option **DEFA2D**, for any ANALOG node connected directly to an HDL-A **IN** port, an implicit A2D will be automatically inserted.

Option D2DMVL9BIT

Enables direct connection between HDL-A ports of type **BIT** with mixed-mode nodes connected to Eldo via convertors of type MVL9 (or std_logic).

The type std_logic is an enumerated type. The type **BIT** is a subset of std_logic, so a conversion table is required as follows.

Table 10-6. Option D2DMVL9BIT effect

A2D mode std_logic	HDL-A PORT bit type
‘0’, ‘L’	‘0’
‘1’, ‘H’	‘1’
“others”	ignored

.AC

AC Analysis

Parameter-Driven Analysis

```
.AC TYPE nb fstart fstop [SWEEP DATA=dataname] [UIC] [MONTE=val]
.AC TYPE nb fstart fstop [SWEEP parameter_name TYPE nb start stop]
+ [UIC] [MONTE=val]
.AC TYPE nb fstart fstop [SWEEP parameter_name start stop incr]
+ [UIC] [MONTE=val]
```

Data-Driven Analysis

```
.AC DATA=dataname [SWEEP DATA=dataname] [UIC] [MONTE=val]
.AC DATA=dataname [SWEEP parameter_name TYPE nb start stop]
+ [UIC] [MONTE=val]
.AC DATA=dataname [SWEEP parameter_name start stop incr] [UIC] [MONTE=val]
```

List-Driven Analysis

```
.AC LIST {list_of_frequency_points}
+ [SWEEP DATA=dataname] [UIC] [MONTE=val]
.AC LIST {list_of_frequency_points}
+ [SWEEP parameter_name TYPE nb start stop] [UIC] [MONTE=val]
.AC LIST {list_of_frequency_points}
+ [SWEEP parameter_name start stop incr] [UIC] [MONTE=val]
```

Adaptive Analysis

```
.AC ADAPTIVE tolerance_value fstart fstop
```

The **.AC** command activates the small signal analysis which computes the magnitude and phase of output variables as a function of frequency. The simulator first computes the circuit DC operating point with user specifiable initial conditions; thereafter the behavior of the linearized circuit is computed for a sine input of user specifiable amplitude, phase and frequency range.

Unit amplitude and zero phase gives the circuit transfer function. The inputs are specified via voltage or current sources which have an AC parameter.

The AC results on circuits containing FNS can be altered by the insertion of a **.PZ** statement in the netlist. The reason being that the **.PZ** statement implies the writing of FNS equations in such a way that is not as stable as the Eldo native FNS equation. However, this effect can be seen only at very high frequencies.

Whenever a **.AC** command, a **.TRAN** command, and a **.OP** command containing time specifications are in the *.cir* file, then an AC analysis will be performed at each of these time points. Please see the description of AC in the middle of a .TRAN for further details.

This command can also be used to perform a sweep on a circuit parameter or device.

The amount of information in the *.chi* file relating to operating point can be limited using the option **PRINT_ACOP**, see "PRINT_ACOP=0|NO|1|YES" on page 1007.

An adaptive variant of AC analysis can be used when there are ports in the design to generate an accurate touchstone file. Eldo will then adjust the frequency in order that sharp peaks in the AC response are not missed.

Negative frequencies can be specified, but only with the **LIN** or **LIST** parameters. For example: **.AC DEC** 10 -1e9 1e9 will result in a warning and –1.0e9 will be changed into 1.0 with a warning. **.AC LIN** 100 -1e9 1e9 is accepted.

Note

The results of AC analysis runs are output using **.PRINT** and **.PLOT** commands.

Parameters

- TYPE

  Can be one of the following:

  DEC

  Keyword to select logarithmic variation.

  OCT

  Keyword to select octave variation.

  LIN

  Keyword to select linear variation. Negative frequencies are allowed with this specified.

  POI

  Keyword to select a list of frequency points. **POI** is the same as **LIST** except that **POI** expects the number of points (NB) to be specified as it's first argument.

  INCR

  Can only be used when the **SWEEP** keyword is preceding it. See also the **INCR** specification in "Sweep Parameters" on page 541.

- **DATA**=dataname

  Used in conjunction with the **.DATA** command. The dataname parameter should be specified using the **.DATA** command. Please refer to ".DATA" on page 585 for more information.

- **LIST**

  Keyword to select a list of frequency points. Negative frequencies are allowed with this specified.

- `nb`

  Number of points per decade or octave or points over the range from `fstart` to `fstop`. This is determined by the preceding keyword.

- `fstart`

  Start frequency in Hertz. This can be specified as a parameter or as an expression.

- `fstop`

  Stop frequency in Hertz. This can be specified as a parameter or as an expression.

- `list_of_frequency_points`

  List of frequency points which can be specified as parameters or as an expression. Values of frequencies must be separated by spaces. When the **`LIST`** keyword is specified, the values must be in increasing order, the order can be increasing or decreasing with the **`POI`** keyword specified.

- **`UIC`**

  Use initial conditions. If specified, no DC analysis is performed before the AC analysis. Instead the circuit may be initialized using **`.RESTART`** or **`.USE`**.

  If you have in your input deck both of the following commands: **`.RESTART`** `file_name`, where `file_name` comes from the results of a TRAN simulation, and if there is a **`.AC`** `...` **`UIC`** command, then the **`.RESTART`** will be interpreted as:

  ```
  .USE file_name OVERWRITE_INPUT
  ```

Refer to "Usage of .RESTART and .AC" on page 855 for more information.

- **`MONTE`**`=val`

  Monte Carlo analysis. Equivalent to **`.MC`** `val`. The syntax allows a different **`MONTE`** value for each run. However, the actual implementation in Eldo does not account for that: it is the last **`MONTE`** value to be specified in the netlist which will be taken into account.

- **`ADAPTIVE`**

  Performs an adaptive AC analysis. Useful if there are ports in the design to have an accurate touchstone file generated. Eldo will adjust the frequency in order that sharp peaks in the AC response are not missed. Otherwise, a large number of points would need to be requested in the AC command in order not to miss such transitions, resulting in very large touchstone files, if any.

- `tolerance_value`

  Specified with an adaptive AC analysis. Tolerance value should be in the range [0.01:0.5]. The smaller the value, the tighter the results. The larger the value, the looser the results.

Sweep Parameters

This section contains **SWEEP** related parameters that are previously unspecified in the Parameters section.

- **SWEEP**

  Specifies that a sweep should be performed on a parameter or device name.

- parameter_name

  Name of the parameter or device name to sweep.

- TYPE

  As defined previously (DEC, OCT, ...). When INCR is specified after the **SWEEP** keyword, then the functionality is as follows:

  INCR

  Increment of the parameter or device name to sweep. When **INCR** is specified as the TYPE parameter, the value which directly follows (nb) is the incrementing value.

- nb

  Number of points per decade or octave or points over the range from start to stop. This is determined by the preceding keyword.

- start

  Start value of the sweep.

- stop

  Stop value of the sweep.

- incr

  Increment of the parameter or device name to sweep.

Note

When **INCR** is specified as the TYPE parameter, the value which directly follows (nb) is the incrementing value. If **INCR** is not specified, the incrementing value (incr) must be placed after the start and stop values.

Examples

```
.ac dec 10 1 1g sweep r2 INCR 10 50 500
.ac dec 10 1 1g sweep r2 50 500 10
```

The two lines above are equivalent. Either can be used to specify a AC analysis from 1Hz to 1GHz with 10 analysis points per decade. The sweep specification will force Eldo to carry out an AC analysis on each value of **r2** starting at 50Ω and stopping at 500Ω with an incrementing value of 10Ω.

```
vin 2 0 ac 0.5
r1 2 3 5k
c3 3 0 0.1p
.ac dec 10 10e+4 10e+8
.plot ac vdb(3)
```

Specifies an AC analysis from 10×10^4Hz to 10×10^8Hz with 10 analysis points per decade. The AC input parameters are defined using the **ac** parameter in the voltage source definition. The results of the AC analysis are plotted in dB for the voltage at node 3 of the circuit in the limits specified in the **.ac** command.

```
v1 1 0 dc 5 ac 0.2
r1 1 2 1k
c1 2 0 1p
.ac oct 8 10e+6 10e+8
.plot ac vdb(2)
```

Specifies an AC analysis from 10×10^6Hz to 10×10^8Hz with 8 analysis points per octave. The input parameters are defined using the **ac** parameter in the voltage source definition.

The results of the AC analysis are plotted for the voltage in dB at node 2 of the circuit within the limits specified in the **.ac** command.

Examples of this type of analysis can be found in "Tutorials" on page 1341.

Parameters are allowed in AC analysis commands as shown in the following example:

```
.param p1 = 1e9
.ac dec 10 1 p1
```

The following example shows the use of an adaptive AC analysis:

```
* S-par extraction

l1 1 1x 1e-7
r1 1x 2x 10
c1 2x 0 3e-12

T1 2x 0 3x 0 Z0=50 Td=1n

c2 3x 0 6e-12
l2 3x 4x 2e-7
r2 4x 2 8

T2 2x 0 5x 0 Z0=40 Td=1.5n

Vp 5x 0 0
c11 1 0 10p
c22 2 0 10p

L3 3x 6x 2e-7
R3 6x 5x 2

I1 0 1 iport=1 rport=50
```

```
I2 0 2 iport=2 rport=50

.extract ac yval(sdb(1,1),1.45e8)

.Ffile S sb1.s2p Hz RI
.plot ac sdb(1,1)
.ac adaptive 0.05 1e5 1e11
```

You will see from the simulation results that Eldo computes some pole-zeros otherwise not seen with a standard AC analysis.

AC in the middle of a .TRAN

Whenever a **`.AC`** command, a **`.TRAN`** command, and a **`.OP`** command containing time specifications are in the *.cir* file, then an AC analysis will be performed at each of these time points.

In the following example Eldo will perform AC, NOISE and TRAN analyses at time 0, time 5n, and time 7n:

```
.AC DEC 10 1 1e9
.NOISE v(5) vin 70
.TRAN 1n 20n
.OP 5n 7n
```

Please see ".NOISE" on page 744, ".OP" on page 755 and ".TRAN" on page 911 for more details on these commands.

In case there are some **`.EXTRACT`** commands related to AC analyses, extract information will be returned for each AC analysis.

When the output is cou format, an extra file, *<circuit>_tac.cou* is created that holds the results of those AC simulations which are performed from within a transient simulation.

When the output is JWDB, a folder TRAN is created with several AC folders inside, for example AC_1, AC_2.

Limitations:

- AC results performed from within a transient simulation will not be dumped in any of the GWl, PSF, and WSF files.
- If there is a **`.TEMP`** or a **`.STEP`** command in the input file, the *.ext* file which normally gets created will not be created in this instance.
- Information related to **`.EXTRACT SWEEP`** will not be available.
- Monte Carlo (**`.MC`**) enabled if **`.MC ALL`** is specified, Worst-Case (**`.WC`**), or transient-noise (**`.NOISETRAN`**) simulations are disabled.

.ADDLIB

Insert a Model or Subcircuit File

```
.ADDLIB N DIR_NAME
```

This command is used to search a directory for files with file extensions *.mod* and *.ckt* (or *.sub*) and include in the circuit description (*.cir*) file the contents of those files whose names correspond to model and subcircuit definitions (respectively) referenced and not defined in the *.cir* file. The **.ADDLIB** command includes a parameter to control the order in which directories are searched.

> **Note**
> When the **.ADDLIB** command is used with binning models, Eldo searches for all the files <model_name>xxxx.mod inside the directories specified inside **.ADDLIB** commands.

The following items need to be stressed for upper and lower case character naming rules conventions:

- A filename must be in all upper or all lower case and appended with a lower-case `.mod` or `.ckt` or `.sub`.

  `.mod` for models only
  `.sub` or `.ckt` for subcircuits only

- Files with upper-case are searched first. For example:

  `My_2N2222a.mod` (not searched)
  `MY_2N2222A.mod` (searched first)
  `my_2n2222a.mod` (searched last)

- For subcircuits: `.ckt` extension is searched first. `.sub` is searched last. For example:

  `op11A.ckt` (not searched)
  `OP11A.ckt` (searched first)
  `op11a.ckt` (searched second)
  `OP11A.sub` (searched third)
  `op11a.sub` (searched last)

Parameters

- `N`

  An integer between 1 and 6, allocating a priority to the directory search command. When several **.ADDLIB** commands are included in a circuit description file, then `N` may be used to determine the order in which the directories are searched. The number 1 allocates the highest priority.

- `DIR_NAME`

  Name of the directory to be searched. The names of the files have to be the same as the subcircuit or model names specified in the netlist.

Example

```
mn1 a b c d mna w=w l=l
   ...
.addlib 2 /users1/examples/
.addlib 1 /users1/models/
```

Specifies that all models and subcircuits missing in the input netlist should be searched for and extracted from *.mod*, *.ckt* or *.sub* files contained in the directories `</users1/models/>` and `</users1/examples/>` in this order. The file containing the model `mna` has to be named *mna.mod*.

.AGE

Age Analysis

```
.AGE [TAGE=value] [TUNIT=year|month|day|hour|min|sec] [NBRUN[S]=value]
+ [LIN[={YES|ON|1}|{NO|OFF|0}] | LOG[={YES|ON|1}|{NO|OFF|0}]]
+ [LOGMODE_MINEXP=value]
+ [TSTART=value] [TSTOP=value] [TWINDOW={(a1,b1) (an,bn)}]
+ [MODE= sim | load | save | MCload | blockload]  [AGELIB=file_name]
+ [AGEALL[={YES|ON|1}|{NO|OFF|0}]]
+ [RESTRICT_MC={YES|ON|1}|{NO|OFF|0}]
+ [ASCII[={YES|ON|1}|{NO|OFF|0}]]
+ [COMPUTE_LAST[={YES|ON|1}|{NO|OFF|0}]]
+ [PLOT={FRESH_FINAL|ALL}]
+ [AGEDSIM={YES|ON|1}|{NO|OFF|0}]
+ [PRINT_CONFIGURATION={YES|ON|1}|{NO|OFF|0}]
+ [STRESS_LIST={YES|ON|1}|{NO|OFF|0}]
+ [STRESS_SORT_NBMAX=value]
+ [STRESS_SORT_REL=value]
+ [STRESS_SORT_ABS=value]
+ [STRESS_LIST_FILE=file_name]
+ [STRESS_LIST_SPLIT_MOS={YES|ON|1}|{NO|OFF|0}]
+ [STRESS_SORT_ORDER=ASCENDING|DESCENDING]
+ [STRESS_SORT_DELTA=extract_label]
+ [DELTA_VDS=value]
+ [DELTA_VGS=value]
+ [DELTA_VBS=value]
+ [USER_WARNING={YES|ON|1}|{NO|OFF|0}]
```

This command activates the Age analysis to test the reliability of a netlist.

For the complete description of Age analysis and information on all reliability commands, see the separate chapter Reliability Simulation.

.AGE_LIB

Define Functions For Reliability

```
.AGE_LIB
+ FILE=library_name
+ FNC_PREFIX=fnc_prefix
+ LEVEL=level1[{, level2, ..., leveln}]
+ [SHARED_PATH=yes|no]
+ [PFDIR=library_path]
```

This command defines a set of functions which describe a reliability process for MOS models.

For the complete description of Age analysis and information on all reliability commands, see the separate chapter Reliability Simulation.

.AGEMODEL

Reliability Model Parameter Declaration

```
.AGEMODEL MODEL=model_name [parameter=value]
```

This reliability analysis command allows the user to specify the model parameters related to reliability calculations.

For the complete description of this command and information on all reliability commands, see the separate chapter Reliability Simulation.

.ALTER

Generalized Re-run Facility

```
.ALTER [LABEL]
[ELEMENT]
[SUBCKT]
[COMMAND]
[COMMENT]
.ALTER | .END
```

Used to re-run Eldo with a modified netlist. All Eldo statements between the **.ALTER** command line and either the next **.ALTER** or **.END** command are back-substituted in the original netlist except for certain commands (see below) which are added to the netlist rather than back-substituted.

The system searches for a match of the component, model, subcircuit or command names in the original netlist and when a match is found the new line is substituted for the original. When no match is found, the new line is simply added, allowing modification of the topology of the circuit.

The following Eldo commands are always added to the netlist with no substitution being attempted.

.PRINT, .PLOT, .CONSO, .CONNECT, .EXTRACT, .GLOBAL, .INCLUDE, .OP, .OPTION, .PARAM, .SENS

If a **.EXTRACT** command is present inside a **.ALTER** command, an *EXT* folder will be created in the *.wdb* file on the condition that the **.EXTRACT** statement remains unchanged for each **.ALTER** file. If this condition is not met, the *EXT* folder is not created. See ".EXTRACT" on page 637.

In a series of **.ALTER** commands, all use the initial netlist as the reference to be altered, not the result of the previous **.ALTER** command.

If one of the **.ALTER** commands produces an error of non-convergence, Eldo will proceed with the next **.ALTER** command.

The *circuit_name.chi* ASCII log file contains a trace of all modifications made. The user may annotate the modification trace label string printed in the *.chi* file using the following syntax:

```
.ALTER <label_string>
```

Use **ALTER_NOMINAL_TEXT** to attach a label to the nominal run. The label used is the same as that specified in the command, for example: **.ALTER** `label`. This label is printed in the *.aex* file or attached to the JWDB waveforms.

Use option **ALTER_SUFFIX** to switch the naming convention for swept waves used in **.ALTER** statements between: *xxx* and *xxx_alter:XX*.

For JWDB output, the alter index number can be viewed when highlighting waveforms in the EZwave Waves window.

Note

The **.ALTER** command may not be used in conjunction with the **.ADDLIB** command. In order to use **.ALTER** with the **.LIB** command, the **KEY** parameter in the **.LIB** is recommended. An alternative is to use **.ALTER** with the **.LIB** command without the **KEY** parameter provided a **.DEL LIB** command is added for each library to be replaced.

.DEL LIB can be used in the **.ALTER** section. The library name specified in the **.DEL LIB** command will be removed from the nominal description.

.MPRUN can be used to take advantage of multi-processor machines for the **.ALTER** command. **.ALTER** has the highest priority when used with **.MPRUN**. Please see ".MPRUN" on page 729 for further information.

Error handling of multiple run netlists can be managed with option **STOPONFIRSTERROR**. When set to 2, the first error stops the simulation even if the netlist file contains **.ALTER** statements for multiple simulation runs. When set to 1, if the netlist file contains **.ALTER** statements, Eldo will stop on the first **.ALTER** that has an error, but continue with the remaining **.ALTER** statements.

To replace one file by another one when using the Eldo re-run facility, use the option **ALTINC**. This forces Eldo to replace the first **.INCLUDE** statement found in an input netlist by the first **.INCLUDE** statement found in the **.ALTER** section of the netlist.

By default, Eldo substitutes the **.STEP** commands in the main netlist by those found in **.ALTER** statements. To append **.STEP** commands in **.ALTER** statements, instead of substituting, use option **ALTER_ADDSTEP**.

.PARAM statements present in a **.ALTER** section are systematically added at the end of the netlist being created for the current ALTER run. Eldo will not attempt to replace the parameter name (as was the case in Eldo versions prior to v6.7_1). This can lead to backward compatibility issues if ALTER was used to replace parameters inside **.SUBCKT** statements, which is not the usual case. If the parameter is defined at the top, outside of the subcircuit, **.PARAM** replacement works without any backward compatibility issues, since the last specification prevails; this is the usual case.

Related Options

"ALTINC" on page 951, "ALTER_ADDSTEP" on page 951, "ALTER_NOMINAL_TEXT" on page 998, "ALTER_SUFFIX" on page 998, "STOPONFIRSTERROR=1|2" on page 960, "WRITE_ALTER_NETLIST" on page 1008.

Examples

```
alter command example
r1 1 2 1k
r2 2 0 1k
c1 2 0 1n
.ac dec 10 200 1000meg
vin 1 0 ac 1
.plot ac vdb(2) (-90, 0)
.alter
r1 1 2 10k
c1 2 0 100p
.ac dec 15 200 1500meg
.end
```

Specifies two simulation runs. Both perform an AC analysis but certain component values, the number of points and stop frequency of the AC analysis are changed using the **.alter** command for the second run.

The example below demonstrates the use of the **KEY** parameter in conjunction with the **.ALTER** command.

```
...
.lib key=K1 /work/bip/mymod typ
.lib key=K2 /work/mos/mymod typ
...
.alter
.lib key=K2 /work/mos/mymod best
.alter
.lib key=K2 /work/private/mymod typ
.end
```

This command sequence causes three simulations to be performed always with library `/work/bip/mymod typ`.
In the first simulation, library `/work/mos/mymod typ` is used.
In the second simulation, library `/work/mos/mymod best` is used.
In the third simulation, library `/work/private/mymod typ` is used.

Note

If there is an error in a **.ALTER** command, it will be skipped and the next **.ALTER** command will be used.

The following example shows option **ALTER_SUFFIX** can be used to switch the naming convention for swept waves used in **.ALTER** statements between: *xxx* and *xxx_alter:XX*. Run the circuit with and without the option and look at the difference in results.

```
*.option alter_suffix=0

.param VTEST1=1
vdc 1 0 dc VTEST1
vin 2 0 dc 0
v2 4 0 dc 1

etest 3 0 value={mult*(2-v(1,2)*v(1,4))}
```

```
r1 3 0 rval

.dc vin 0 2 0.1
.step param VTEST1 0.5 1.5 0.1

.plot dc i(r1)

.extract label=imax max(i(r1))
.extract label=imin min(i(r1))

.defwave sweep TEST=meas(imax)

.param RVAL=1k
.param MULT=1
.alter first
.param RVAL=1.3k
.param MULT=1.3
.alter second
.param RVAL=0.7k
.param MULT=1.3
.alter
.param RVAL=0.7k
.param MULT=0.7
.alter fourth
.param RVAL=1.3k
.param MULT=0.7

.end
```

The following example shows how to replace one included file by another one when using the Eldo re-run facility, with the option **ALTINC**. The *new_models* file will be included in the re-run simulation instead of *models*.

```
.include models
.option altinc
r1 1 0 rval
v1 1 0 dc 1
.extract dc i(r1)
.dc
.alter
.include new_models
```

The following examples show the handling of **.PARAM** statements in **.ALTER** sections. The behavior changed in Eldo version v6.7_1.

- Backward compatibility issue if ALTER is used to replace **.PARAM** inside **.SUBCKT** statements, which is not the usual case.

```
.subckt foo 1 2
.param p1 = 1
r1 1 2 p1
.ends

i1 1 0 1
x1 1 0 foo
.op
```

```
.alter
.param p1 = 2
.end
```

Here, the netlist corresponding to the first ALTER will now be:

```
.subckt foo 1 2
.param p1 = 1
r1 1 2 p1
.ends

i1 1 0 1
x1 1 0 foo
.op
.param p1 = 2
.end
```

This means that the value of r1 will remain at 1. In Eldo versions prior to v6.7_1, p1=1 would have been replaced by p1=2 even inside the subcircuit.

- No backward compatibility issue

```
.subckt foo 1 2
.param p1 = 1
r1 1 2 p1
.ends

.param p2 = 1
i1 1 0 p2
x1 1 0 foo
.op
.alter
.param p2 = 2
.end
```

Here, for the ALTER run, the value of i1 will be 2, as expected.

-compat flag

Usually in Eldo, **`.ALTER`** restarts from the original netlist, however, with the `-compat` flag set, **`.ALTER`** is cumulative. For example:

```
r1 1 0 1
r2 2 0 1
.alter 1
r1 1 0 2
.alter 2
r2 2 0 2
```

If Eldo is running with the `-compat` flag set, the second “alter” simulation will be done with both `r1` set to 2 (inheritance from later number 1), and `r2` set to 2.

If the `-compat` flag is omitted, then the second “alter” simulation will be run with `r1` set to 1 (original netlist) and `r2` set to 2.

.BIND

Configure Spice Descriptions

```
.BIND INST=inst_name [EXCEPT=inst_name]
| FROM_SUBCKT=Eldo_subckt_name | FROM_MODEL=HDL_model_name
  TO_SUBCKT=new_Eldo_subckt_name
  [FILE=<file_name> VARIANT=<variant_name>]
| TO_MODEL=new_HDL_model_name
 [MAPPING=assoc_file_name | DEFAULT_MAPPING=by_name|by_position]

.BIND INST=inst_name [EXCEPT=inst_name]
  FROM_SUBCKT=Eldo_subckt_name | FROM_MODEL=HDL_model_name
  TO_SUBCKT=new_Eldo_subckt_name
  [FILE=<file_name> VARIANT=<variant_name>]
| TO_MODEL=new_HDL_model_name
 [MAPPING=assoc_file_name | DEFAULT_MAPPING=by_name|by_position]
```

Change one SPICE description (for example, one automatically generated by any schematic tool) to another equivalent description without editing the description itself (or netlisting it again).

The **.BIND** command is used to substitute any SPICE subcircuit by either another SPICE subcircuit or a behavioral model (VHDL, VHDL-AMS, Verilog or Verilog-AMS).

For more information on **.BIND**, see Configuring SPICE Descriptions of the *ADVance MS User's Manual*.

The **.BINDSCOPE** command can be used in conjunction with the **.BIND** command to specify either the *full* or *part* of the hierarchical path to the instance(s) that will be replaced.

Note

The path specified on the **.BINDSCOPE** command specifies the hierarchical path for *all* **.BIND** commands, therefore the path on the **.BIND** command must be relevant to the path specified on the **.BINDSCOPE** command.

Parameters

- **INST**=inst_name

  Specifies the name of the instance(s) that will be replaced with an equivalent SPICE subcircuit or behavioral model. This can be restricted to specific instances of SPICE subcircuits or behavioral models by specifying either **FROM_SUBCKT**=... or **FROM_MODEL**=....

- **EXCEPT**=inst_name

  Specifies the name of the instance(s) that will *not* be replaced with an equivalent SPICE subcircuit or behavioral model.

- **FROM_SUBCKT**=Eldo_subckt_name

  If the Eldo subckt name is not provided, it is equivalent to any subcircuit (in this case, only the instance name is used). If not specified, all the subcircuits will be replaced by the specified subcircuit or HDL model. This can be restricted by specifying **INST**=inst_name.

- **FROM_MODEL**=HDL_model_name

  If model name is not provided, it is equivalent to any model (in this case, only the instance name is used). If not specified, all the HDL models will be replaced by the specified subcircuit or HDL model. This can be restricted by specifying **INST**=inst_name.

- **TO_SUBCKT**=new_Eldo_subckt_name

  The Eldo subcircuit name that will replace the existing SPICE or behavioral model(s). The Eldo subcircuit can be replaced with another Eldo subcircuit of the same name from a different library. To do this use the arguments **FILE**=file_name and **VARIANT**=variant_name.

- **FILE**=file_name

  Name of the file that contains the library.

- **VARIANT**=variant_name

  Name of the variant.

- **TO_MODEL**=new_HDL_model_name

  The behavioral model name that will replace the existing SPICE or behavioral model(s).

- **MAPPING**=assoc_file_name

  Specifies the interface association file to be applied to the port mapping between the model to replace and the new model. This is optional. By default, a mapping by position is used when this file is not provided.

- **DEFAULT_MAPPING**= by_name|by_position

  Defines the port mapping between the model to replace and the new model. The ports can be mapped either by name or by position. This is optional.

For the instance and subcircuit names, SPICE wildcards as listed in Table 10-7 can be used:

Table 10-7. Wildcard Characters

Character	Description
*	Matches any sequence of characters, including the hierarchy separator "."
?	Matches any single character.
[abcd]	Matches any character in the specified set.
[-abcd]	Matches any character in the specified range, for example [A-Z] matches all alphabetical characters.

Table 10-7. Wildcard Characters

Character	Description
[!abcd]	Matches any character not in the specified set, for example [!0-9] matches all characters which are not digits.

Take care when combining wildcards. If the name used in a .BIND contains square brackets, "[" and "]", these are interpreted as special characters when the wildcard characters "*" or "?" are used. According to UNIX convention, these special characters may be overridden by using the delimiter backslash character "\" which removes the special function of the character that immediately follows it. For example, the following only returns names that end with XVCO1, because, according to UNIX convention, *.XVCO[1] means "all names ending with "XVCO" and followed by the character "1"":

```
.BIND inst=*.XVCO[1] from_subckt...
```

Wildcards may be used either for the instance name or the subcircuit name that will be replaced (`inst` or `from_subckt`), but not for the **`.MODEL`** card or subcircuit that is used in replacement (`to_subckt` or `to_model`). Some examples:

```
adc?top     can be adc1top, adcztop but not adc10top.
adc*top     can be adc1top, adcztop, adc10top and adc12.b23top.
adc[0-9]top can be adc1top but not adcztop.
```

Examples

Suppose that we have a SPICE description called *top.cir*. To substitute all instances of subcircuit *inverter* that are part of the sub-instance called *X1.X2* by the Verilog model *inverter* compiled in the working library, a new file called *new-top.cir* can be written as follows:

```
* file new-top.cir
.INCLUDE  top.cir
* substitution model declaration
.MODEL  new_inverter  macro  lang=verilogams
+     mod=inverter  param: delay=10ns
* substitution command
.BIND  inst=x1.x2.*
+     from_subckt=inverter  to_model=new_inverter
```

The following example is the same as the previous example however, the **`.BINDSCOPE`** command is used in conjunction with the **`.BIND`** command to define the path to the instances that will be replaced:

```
* file new-top.cir
.INCLUDE  top.cir
* substitution model declaration
.MODEL  new_inverter  macro  lang=verilogams
+     mod=inverter  param: delay=10ns
* substitution command
.BINDSCOPE PATH=x1.x2
.BIND  inst=*
```

```
+       from_subckt=inverter  to_model=new_inverter
```

.BINDSCOPE

Define local scope for .BIND

```
.BINDSCOPE PATH=scope_path
```

Specifies the hierarchical path to the instance(s) that will be replaced using the .BIND command. The path specified will apply to all **.BIND** commands in the design. Therefore, the path specified on the **.BIND** command must be relevant to the path specified on the **.BINDSCOPE** command.

Parameters

- **PATH**=scope_path

  Part of or the *full* hierarchical path to the instance(s) that will be replaced using the **.BIND** command.

Note

A warning will be generated if multiple **.BINDSCOPE** commands are defined in a netlist. Only the last occurrence of the command will be used.

.CALL_TCL

Call Tcl Function

```
.CALL_TCL [TRAN|AC|DC|...]
+ WHERE=START|START_OF_RUN|END_OF_RUN|END
+ [PLOT=[YES|NO|0|1]] [PLOT_TYPE=[TRAN|AC|DC|...]]
+ [LABEL=alias_name] tcl_function_call
```

This command is used to perform a single call to a Tcl function.

This allows you to dynamically manage User Defined Functions (UDF) written in Tcl language. The functions of the post-processor library and other commands are available through the Tcl interface.

> For loading a Tcl file containing functions into Eldo's Tcl interpreter, see the command ".USE_TCL" on page 929. For further information on both commands, see the Post-Processing Library chapter of this manual.

Parameters

- `WHERE`

  Specify when Eldo must call the tcl function:

  `START`

  Only one call at the very beginning of the simulation.

  `START_OF_RUN`

  One call at the beginning of each run.

  `END_OF_RUN`

  One call at the end of each run.

  `END`

  One call at the very end of the simulation.

- `PLOT`

  If the Tcl function returns a wave, this keyword asks Eldo to dump the wave in the output file.

- `PLOT_TYPE`

  Informs Eldo which type of waveform to expect. This is used when the result of the Tcl function is a waveform which must be plotted (PLOT=YES set on the command).

- `LABEL`

  This name will be used to plot the wave in the output file.

- `tcl_function_call`

  The Tcl function. A function can take any kind of arguments: waves, numbers and keywords. These keywords are: `IRUN`, `ICARLO`, `IALTER` which respectively represent the index of the current step, Monte Carlo run, and **`.ALTER`**. `NBRUN`, `NBCARLO`, `NBALTER` which represent the number of steps, Monte Carlo runs, and **`.ALTER`**.

Example

```
.call_tcl tran WHERE=END_OF_RUN
+ MY_FUNCTION(v(1),2.0,irun)
```

.CHECKBUS

Check Bus Values

```
.CHECKBUS BNAME [VTH[1]=VAL [VTH2=VAL]] [BASE=DEC|OCT|BIN|HEX] [LOCK=1]
+ [REPORTX=0|1] TN VAL {TN VAL}
.CHECKBUS BNAME [VTH[1]=VAL [VTH2=VAL]] [TSAMPLE=VAL] [TDELAY=VAL]
+ [BASE=DEC|OCT|BIN|HEX] [LOCK=1] [REPORTX=0|1] PATTERN BITS {BITS}
```

Checks that the value of the bus has the value `VAL` at time `TN`. If an error occurs, the list of the errors are printed inside the *.chi* file. It is also possible to specify expected values using the **`PATTERN`** function.

Parameters

- `BNAME`

  Bus name.

- `TN`

  Time in seconds at which the bus signal should be equal to `VAL` volts.

- `VAL`

  Bus signal voltage level at time `TN`.

- **`PATTERN`** `BITS {BITS}`

  Bus signal voltage levels representing the **`PATTERN`** source. For instance, the 4th `VAL` specified is the voltage level at time `TSAMPLE`×4 + `TDELAY`. Integer values can be specified, a bus value specifying `base=bin pattern 101` has the same effect as specifying `base=dec pattern 5`.

- **`VTH`**`[`**`1`**`]=VAL`

  The `VTH` parameters are required by Eldo to compute the HEX (or DEC, OCT, or BIN) value on the bus from the analog value inside Eldo. Then, it compares this value with the value expected and displays the error if they are not the same.

- **`VTH2`**`=VAL`

  This can be used to plot the indeterminate value as shown below:

  When only **`VTH1`** is given:
  If value < **`VTH`** then logic state 0.
  If value > **`VTH`** then logic state 1.

  When both **`VTH1`** and **`VTH2`** are given:
  If value < **`VTH1`** then logic state 0.
  If **`VTH1`** < value < **`VTH2`** then state X.
  If value > **`VTH2`** then logic state 1.

- **`BASE`**

  Keyword indicating that the bus signal number system is to be defined.

  `OCT`

Keyword indicating that bus signals are defined in octal.

DEC

Keyword indicating that bus signals are defined in decimal. Default.

BIN

Keyword indicating that bus signals are defined in binary.

HEX

Keyword indicating that bus signals are defined in hexadecimal.

- **TSAMPLE**=VAL

  Time spent at 1 or 0 **PATTERN** value.

- **TDELAY**=VAL

  Delay before the **PATTERN** series is started.

- **LOCK**=1

  Forces Eldo to compute specified time points. By default, **LOCK** is 0.

- **REPORTX**=0|1

  Print checkbus warnings for undefined X states. By default, **REPORTX** is 0. Eldo considers that X states taken by the bus or given in a checkbus list mean that every state is correct for this time point. Thus the check is passed if the bus is X and the check value is 0 or 1.

Related Options

MAX_CHECKBUS, see "MAX_CHECKBUS=ALL|val" on page 1003.

Examples

```
.CHECKBUS OUT_BUS base=bin vth=2.5
+5000PS  01111
+10000PS 01011
+20000PS 01011
.CHECKBUS OUT_BUS_2 vth=1 vth2=4 base=bin
+ 1000ps 01
+ 7000ps x
+ 12000ps 11
+ 15000ps 01
+ 19500ps 01
```

Refer to ".PLOTBUS" on page 827 for more information.

The following three checkbus lines are equivalent. They use three different bases to specify the expected value on the bus.

```
.checkbus my_bus vth=2.5 base=hex 15m 15 25m 22 45m 91
.checkbus my_bus vth=2.5 base=dec 15m 21 25m 34 45m 145
.checkbus my_bus vth=2.5 base=oct 15m 25 25m 42 45m 221
```

The following example shows how it is possible to specify expected values using the **PATTERN** function:

```
.checkbus my_bus vth=2.5 TSAMPLE=10m TDELAY=5m base=hex
+ PATTERN 10 15 22 22 91
```

is equivalent to:

```
.checkbus my_bus vth=2.5 base=hex
+ 5m 10 15m 15 25m 22 35m 22 45m 91
```

Refer to the “Pattern Function” on page 332 for more information.

The following example shows how the LOCK parameter can be used:

```
.CHECKBUS S VTH1=0.5 VTH2=4.5 BASE=HEX LOCK=1
+  15N 0  35N 1  55N 2  75N 3
+  95N 1 115N 2 135N 3 155N 0
+ 175N 2 195N 3 196.5n 1 215N 0 235N 1
+ 255N 3 275N 0 295N 1 315N 2
```

Eldo will be forced to compute the timepoints 15n, 35n, 55n, and so on, and the result of the checkbus will produce the following for any accuracy setting:

```
at time 1.965000e+02 ns bus S is 3 and should be 1
```

Without LOCK set, or when LOCK=0, the time reported may vary depending on the circuit and tolerance setting, that is, the nearest point to 1.965000e+02 ns as computed by Eldo. When LOCK=1, the time reported will be 1.965000e+02 ns for any accuracy setting.

.CHECKSOA

Check Safe Operating Area Limits

```
.CHECKSOA [ANALYSIS] [TSTART=val [TSTOP=val]] [TMIN=val]] [AUTOSTOP]
+ [NOMERGE] [NOLIB] [FILE=filename] [NOXWINDOW]
+ [ENABLE|DISABLE] [SOACODE=range1[{,range_x}]]
+ [INST={list_of_instances}] [SUBCKT={list_of_subckts}] [RUNTMSG]
```

Causes Eldo to check for any violations of the circuit safe operating area (SOA) limits specified via the **.SETSOA** command.

See “.SETSOA” on page 872 for more details.

If an entity has more than one set of the same specifications, but with different boundaries, Eldo merges the specifications by taking the more restrictive constraint. This is the default mode.

You can filter SOA checks to enable/disable based on a code assigned to any SOA check using the **SOACODE** parameter of the **.SETSOA** command.

The targets of the SOA can also be specified with more accuracy to enable/disable the SOA checks on specific instances or parts of the circuit hierarchy.

Parameters

- ANALYSIS

  Optional. By default, it depends on the analysis specified in the netlist. Can be one of the following:

  AC

  Specifies that the checks are required for an AC analysis.

  DC

  Specifies that the checks are required for a DC analysis.

  TRAN

  Specifies that the checks are required for a transient analysis.

  DCSWEEP

  Specifies that the checks are required for a DCSWEEP analysis.

  FOUR

  Specifies that the checks are required for an FFT analysis.

  TMODSST|TSST|FSST

  Specifies that the checks are required for an RF analysis.

- **TSTART**=val

  Specifies start time value when SOA should be checked. Default is the first time point of the TRAN simulation.

- **TSTOP**=val

  Specifies stop time value when SOA should be checked. Default is the last time point of the TRAN simulation.

- **TMIN**=val

  Specifies the value for a minimum time window when SOA should be checked. When this value is set, Eldo will ignore transient SOA violations lasting less than **TMIN** seconds. Default value is 0.

- **AUTOSTOP**

  Forces Eldo to quit immediately if an SOA specification is violated.

- **NOMERGE**

  By default, if an entity has more than one set of the same specifications, but with different boundaries, Eldo merges the specifications by taking the more restrictive constraint. Use **NOMERGE** to take the constraints into account as specified in the netlist. For example:

  ```
  .setsoa label=m1 d m5 vgs=(*,1.36)
  .setsoa label=m2 d m5 vgs=(*,1.37)
  ```

  The second constraint will be ignored, since the constraint labelled `m1` implies constraint `m2`. If the keyword **NOMERGE** is specified on the **.CHECKSOA** command, the two constraints will be taken into account as specified in the netlist.

- **NOLIB**

  Eldo will not consider SOA cards which are imported from **.LIB**. Instead, it will only consider those specified in the *.cir* file or from **.INCLUDE** files.

- **FILE**=filename

  Specifies a file to which SOA results will be written instead of the standard ASCII output file. If no violations occur the message “No SOA violation detected” will be printed in the file.

- **NOXWINDOW**

  Forces Eldo not to write the X-axis window information if the SOA is violated.

- **ENABLE|DISABLE**

  Enable or disable the indicated targets. By default, they are enabled.

- **SOACODE**=range

  A single code or range of codes assigned to SOA checks in **.SETSOA** commands. Based on these codes you can filter SOA checks to enable/disable. The codes should be positive integer quantities. Separate a list of codes by commas. A range of contiguous codes can also be specified using an n1:n2 syntax to designate all codes between n1 and n2. The **SOACODE**, if specified, must be placed immediately following the **ENABLE|DISABLE** parameter. For example:

```
.CHECKSOA TRAN ENABLE SOACODE=1,2 INST=x1*
```

- **INST**

  Restricts (enable/disable) the scope of **.SETSOA** cards to the instances specified by `list_of_instances`. Only **.SETSOA** commands which explicitly reference the instance will be (enabled/disabled) .

- **SUBCKT**

  Restricts (enable/disable) the scope of **.SETSOA** cards to the subcircuit instances specified by `list_of_subckts`. Only **.SETSOA** commands which are in the subcircuit definition or which explicitly reference the instance will be enabled/disabled.

- **RUNTMSG**

  Forces Eldo to display the SOA violations on screen as they are detected during the simulation. This means you do not have to wait till the end of the simulation to see when the violations occur. The violation information displayed on screen is represented differently to what is printed in the *.chi* file. If the `XAXIS` parameter is specified in the **.SETSOA** command then violations which occur within this time span and continue afterwards will only be displayed on screen when the `XAXIS` time span finishes. Violations which start and finish within the `XAXIS` time span will not be displayed.

Example

```
* Restrict SOA to list subckt instances
.model resis res r= 1k dev/gauss=10%

.subckt xsub a b
r1 a b resis 1k
.setsoa E I(R1)=(1.80u,1.90u) !<= This card will be active
*                                 for X1 only
.ends xsub

v1 1 0 pwl(0 0 10n 10)
x1 1 2 xsub
x2 1 2 xsub
r2 2 0 1k

.setsoa D X1.R1 i=(1.80u,1.90u)       !<= This soa is active
.setsoa D X2.R1 i=(1.80u,1.90u) !<= This one is not active
.checksoa subckt=x1

.tran 1n 10n
.extract tran I(X1.R1)
.end
```

In the example above, two cards will be kept: the one in `xsub` corresponding to `X1`; and the top level card **.setsoa** `D X1.R1`.

.CHRENT

Piece Wise Linear Source

```
.CHRENT NODE TN VN {TN VN} [P|F]
.CHRENT NODE (TN VN {TN VN}) FACTN {(TN VN {TN VN}) FACTN} [P|F]
```

Generates a Piece Wise Linear source using straight lines between specified points as described below.

Parameters

- `NODE`

  The source is placed between node `NODE` and ground.

- `VN`

  Value of the source at time ***Ti*** in volts. The source value at intermediate times is provided by linear interpolation.

- `TN`

  Time in seconds, at which ***Vi*** is supplied, where $Ti < Ti + 1$.

- `FACTN`

  Factor that indicates how many times the previous block is repeated.

- **`P`**

  Defines a periodic signal type. Default signal. Optional.

- **`F`**

  Defines a non-periodic signal type. Optional.

Notes

The time declared using the first syntax above is an *absolute* time whereas that declared using the second syntax in the parentheses is *relative*. This makes life easier when dealing with complex waveforms since calculation of absolute times can be very tedious.

For non-periodic curves, the voltage applied to node `NODE` between time 0 and time `T1` is equal to `V1`, with the voltage applied at time `TN` being `VN`. Time `TN` specifies the last time interval.

For periodic curves, the voltage applied to node `NODE` between time 0 and `T1` is `V1`. The period however, is specified as `TN` minus `T1` enabling an initialization phase before starting a periodic signal.

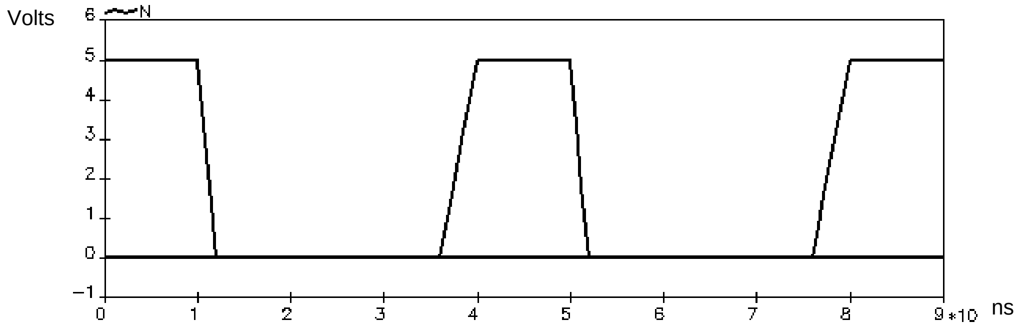
A piece-wise linear signal may be declared both by a **`.CHRENT`** command and a Piece Wise Linear signal on an independent source **`V`**`xx`. The repetition factor, however, may only be declared using the **`.CHRENT`** command.

Examples

```
.chrent n3 0n 5.0 10n 5.0 12n 0 36n 0 40n 5.0 p
```

Specifies a periodic signal applied between node `n3` and ground. It starts at 0s with 5V, stays there for 5ns, falls in 1ns to 0V, stays there for 32ns, then it rises in 2ns to 5V. This signal repeats with a period of 40ns.

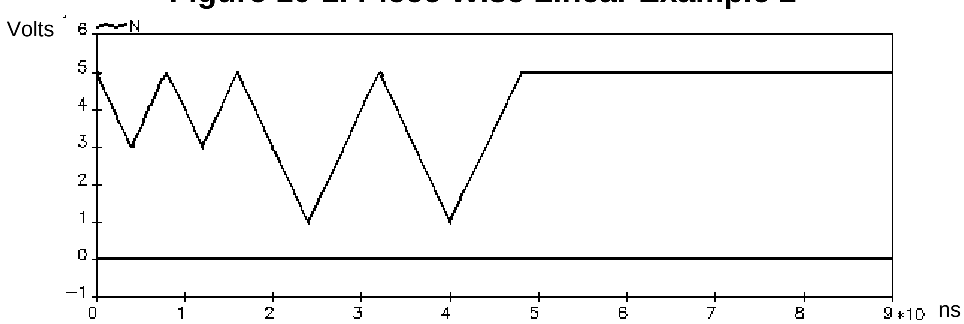
Figure 10-1. Piece Wise Linear Example 1




```
.chrent node1(0 5 4n 3 8n 5)2(0 5 8n 1 11n 5)2
```

This example illustrates a way of describing a function which is periodic for a given period of time, then becomes periodic with a different waveform for another period of time, and so on. The values not in parentheses indicate how often the block is repeated.

Figure 10-2. Piece Wise Linear Example 2



.CHRSIM

Input from a Prior Simulation

```
.CHRSIM IN OUT FILE [TSTART=V1] [TSTEP=V2] [BP=0|1|2]
+ [ZOOMTIME] [FORMAT=WDB] [RUN=VAL] [TYPE=CURRENT|VOLTAGE]
```

The **.CHRSIM** command enables the user to use the output of previous simulations as input to the current simulation. The previous simulation data must have been generated by a DC sweep or a transient analysis. The simulator assigns the output obtained at node `OUT` of the circuit `FILE` to the node `IN` of the current simulation.

.CHRSIM can read in any waveform from a waveform file using the EZwave name. It can handle any waveform sent to any waveform file: currents, Monte Carlo runs, and so on. For example, maximum and minimum waveforms created by a **.MC** analysis (without the **ALL** keyword specified) are recognized by **.CHRSIM** for example:

```
.chrsim x "v(2)_H" e.wdb bp=2
```

This would read in the waveform `v(2)_H` from the output file *e.wdb*, and use it as an input at node `x` in the current simulation. The waveform `v(2)_H` is a maximum result coming from a Monte Carlo run.

Note

There is a difference in the **.CHRSIM** syntax between Eldo and ADMS. In ADMS, the name of the top design must be declared for previously simulated waveforms. For example:

```
.CHRSIM O1 V(O1)        ./inv1 BP=2 TYPE=voltage ! for Eldo
.CHRSIM O1 V(:inv1:O1) ./inv1 BP=2 TYPE=voltage ! for ADMS
```

Parameters

- `IN`

  Name of the input node of the current simulation.

- `OUT`

  Name of the node, or waveform, previously simulated in `FILE`.

- `FILE`

  Name of the circuit previously simulated to be read. This file must be located in the same directory as the file to be simulated, and node `OUT` must have been requested via a **.PLOT** command. If you specify the full name of the file, including extension, the **FORMAT** specification will not be required. By default, if no extension is provided, Eldo will look for a *.cou* file unless **FORMAT** is specified.

- **TSTART**=V1

  Starting position in the *FILE.wdb* file in seconds.

- **TSTEP**=V2

  Selects one point every **TSTEP** points in the *FILE.wdb* file.

- **BP**=0

  Input wave will not impose time points on Eldo. Default.

- **BP**=2

  Eldo will make a time point at each time point found in the input wave.

- **BP**=1

  Intermediate between cases 0 and 2. Eldo will impose a time point if the variation of the input wave is significant enough.

- **ZOOMTIME**

  Reuse the results of a simulation with **.CHRSIM** by modifying the time scale. If the simulation read back corresponds to a TRAN simulation, and if the current simulation is also a transient simulation for which the simulation duration is known (that is, **.TRAN** command present in the netlist), then the simulation time as read in the file will be multiplied by `TSTOP/TSTOP_READ`.

  `TSTOP` is the simulation duration as specified in the **.TRAN** command. `TSTOP_READ` is the simulation duration as read in the 'cou-format' file specified in the **.CHRSIM** command.

- **FORMAT**=**WDB**

  .CHRSIM will retrieve data values from a JWDB file with extension *.wdb*. Not required if you specify the extension in `FILE`.

- **RUN**=VAL

  Specifies which simulation run the results will be used from, for use with multi-run analysis. Default is 1 (first simulation run).

- **TYPE**=**CURRENT|VOLTAGE**

  Specifies whether to create a current or voltage source. Default is voltage.

Examples

```
.chrsim n7 a8 adder.wdb
```

Specifies the output of the circuit `adder` at node `a8` to be applied at node `n7` of the current simulation. Eldo reads the file *adder.wdb* which must be in the same directory.

```
.chrsim i7 o2 mult.wdb tstart=4n tstep=1n
```

Specifies the output of the circuit `mult` at node `o2` to be applied at node `i7` of the current simulation. The signal is applied 4ns in from its starting point at 1ns intervals of the waveform. Eldo reads the file *mult.wdb* which must be in the same directory.

.COMCHAR

Change Comment Character

```
.COMCHAR char {char}
```

This command is used to change the comment character on an input file. Several comment characters can be specified as active at the same time in a single **.COMCHAR** command. Any new **.COMCHAR** command will append the new character to the list of comment characters.

Parameters

- char

  A list of comment characters can be specified. The command accepts any character. Usually, only the $ or the ! characters are used. Unexpected behavior can be obtained when specifying other characters. The default is character '!'. By default, Eldo also allows the '*' character as an inline comment except on **.EXTRACT**, **.DEFMAC**, **.DEFWAVE**, **.MEAS**, **.OBJECTIVE**, and **.PARAM** commands.

Notes

- The first call to **.COMCHAR** overwrites the default comment character, that is, character '!' in Eldo default mode.
- Specifying option **CUMUL_COMCHAR**= 0 returns to the exclusive mode (pre-v6.4 default scheme) where only the last **.COMCHAR** specified is effective. An alternative is to specify command **.RESETCOMCHAR** <list_of_char> which will overwrite the list of comment characters.
- The following warning will be printed when the comment character is $:

  ```
  Warning 1605: Eldo uses the $ character as a comment character.
  There are several ways to use this inside the netlist:
  +   - to specify string parameters ($(FOO))
  +   - to specify environment variables (.include $MY_PATH/foo.inc)
  +   - to call macros defined by a .defmac (.param foo = $sum(2,3))
  + If Eldo returns a parsing error on a line containing such an
  + element, try enclosing it between ' or ", for example:
  '$(FOO)' / .include "$MY_PATH/foo.inc" / .param foo = '$sum(2,3)'
  ```

  Enclose the $(param_name) between single quotes ' in order to use it when the $ is used as a comment character.

Examples

To tell Eldo that the comment character on an input file is '$' rather than the default '!', use the command:

```
.comchar $
```

The example below shows how a list of comment characters can be made active in a single command, and then how the list is appended to with a further command:

```
Title
.COMCHAR $#!
.COMCHAR /
i1 1 0 1 !default comment
r1 1 0 3 $another comment
r2 1 0 3 #another comment
r3 1 0 3 /another comment
.dc
.end
```

.CONNECT

Connect Two Nodes

```
.CONNECT N1 N2
```

The **.CONNECT** command is used to connect the nodes `N1` and `N2` without modifying the circuit description file. Wildcard characters ‘*’ can be specified to make multiple connections with a single command, Eldo will attempt to connect any node name which matches the pattern.

Parameters

- `N1, N2`

  Names of the nodes to be connected.

Examples

```
.connect n7 n5
```

Specifies the connection of the nodes `n7` and `n5`.

```
.connect in<*> 1
v1 1 0 1
r1 in<1> 0 1
r2 in<2> 0 1
r3 in<3> 0 1
.op
.end
```

This example shows wildcards being used to make multiple connections. Eldo will connect nodes `in<1>`, `in<2>`, and `in<3>` to node 1.

.CONSO

Current Used by a Circuit

```
.CONSO VN {VN}
```

The **.CONSO** command computes and displays the average current flowing through the specified voltage source(s) during the simulation period. There is no limit to the number of **.CONSO** commands that may be present in an input netlist.

Parameters

- VN

  Name of the voltage source(s).

Note

This command may only be used in conjunction with a transient analysis.

Examples

```
vdd 100 101 5v
...
.conso vdd
.tran 1ns 100ns
```

Specifies a printed output of the average current flowing through the voltage source vdd over the length of the circuit simulation.

```
v1 n1 0 5
v2 n3 n4 2
...
.conso v1 v2
.tran 1ms 50ms
```

Specifies a printed output of the average current flowing through the voltage sources v1 and v2 over the length of the circuit simulation.

.CORREL

Correlation between Parameters

```
.CORREL [PARAM=]param_list cc=VAL
.CORREL DEV[ICE]=device_list PARAM=param_list cc=VAL
```

Correlations can be specified between parameters during a Monte Carlo analysis. This is done by specifying the relations between the different parameters. Correlation coefficients can be given for groups of two parameters. It can also be given for a list of parameters, in such a case the correlation coefficient will be the same for each couple. For example, in the list a, b, c the couples will be (a, b), (b, c) and (c, a).

It is also possible to specify correlations for each subcircuit instance via parameters which are assigned **DEVX** variation.

For more information see **DEVX** in **.PARAM**.

From the correlated parameters specified via **.CORREL** statements, Eldo retrieves a set of uncorrelated parameters and computes a conversion matrix between the uncorrelated parameters and the correlated parameters. This is done using the PCA method (Principal Component Analysis). Then, it is on these uncorrelated parameters that the MC variations will be performed. Eldo will then compute the new values for the correlated parameters, using the transformation matrix.

Parameters

- **PARAM**=param_list

  List of parameters to be correlated. There is no limit to the number of parameters that can be listed. They can be listed in a number of ways, as follows:

  ```
  .correl a b c cc=val
  .correl param=a b c cc=val
  .correl param=a, b, c cc=val
  .correl param={a b c} cc=val
  .correl param={a, b, c} cc=val
  ```

  When used with **DEVICE** it represents a list of parameters which have **DEVX** specifications.

- **cc**=VAL

  Correlation coefficient. This can be any real number between -1 and +1.

- **DEVICE**=device_list

  List of devices and subckt instances.

Examples

The following command:

```
.CORREL param=p1,p2,p3 cc=0.1
```

is equivalent to the following three **.CORREL** commands:

```
.CORREL p1 p2 cc=0.1
.CORREL p1 p3 cc=0.1
.CORREL p2 p3 cc=0.1
```

The following example shows how parameters with **LOT** variations can be correlated:

```
.param pc=10p LOT=10%
.param rc=1k  LOT=5%
.SUBCKT cmod a b
C1 a b pc
R1 a b rc
.ENDS
X1 4 0 cmod
X2 6 8 cmod

.CORREL param=pc,rc cc=0.9
```

Eldo will generate pseudo-random values for `pc` and `rc` using 0.9 as their coefficient of correlation.

The following example shows how parameters with **DEVX** variations can be correlated:

```
.param pc=10p DEVX=10%
.param rc=1k  DEVX=5%
.SUBCKT cmod a b
C1 a b pc
R1 a b rc
.ENDS
X1 4 0 cmod
X2 6 8 cmod
.CORREL device=x1,x2 param=pc,rc cc=0.1
```

The **.CORREL** command will create a relation between `X1.pc` and `X2.pc` and also between `X1.rc` and `X2.rc` with a coefficient of 0.9.

Note

In the case of correlation on devices, and when the parameter list is not specified, then all parameters with **DEVX** variations which are used in the X instances given in the device list will use the same correlation coefficient **cc**.

.D2A

Digital-to-Analog Converter

```
.D2A [SIM=simulator] eldo_node_name
+ [digital_node_name] [MOD=model_name] [parameters_list]
```

The **.D2A** statement establishes the interface connection between digital solvers and Eldo.

For Analog-to-Digital, please refer to ".A2D" on page 528.

When the digital value driving the "mixed-signal" is logic '1', the voltage on the analog side of the signal is driven to the voltage value defined by the parameter VHI. When the logic level is '0', the analog voltage is driven to that defined by VLO. TRISE and TFALL are used to specify the duration of the linear rise and fall slopes between the voltages VHI and VLO. Values of "zero" for TRISE and TFALL are not allowed since this would represent an infinite energy transition.

The following table shows a global view of the parameters of the **.D2A**:

Table 10-8. .D2A Global Parameters

Name	Description	Default	Units
MODE= BIT | X01 | MVL4 | STD_VSRC			
VHI	Higher voltage value, corresponding to the logical '1' state. If VHIREF is specified the parameter VHI will be ignored.	5	V
VLO	Lower voltage value, corresponding to the logical '0' state. If VLOREF is specified the parameter VLO will be ignored.	0	V
VHIREF	Value of this node is used to set the higher voltage reference, corresponding to the logical state '1'. If specified the value of VHI will be ignored.	-	-
VLOREF	Value of this node is used to set the lower voltage reference, corresponding to the logical state '0'. If specified the value of VLO will be ignored.	-	-
TRISE	Defines the rise time for the voltage source, going from VLO to VHI value	2ns	s
TFALL	Defines the fall time for the voltage source going from VHI to VLO value	2ns	s
MODE= X01Z | STD_LOGIC			
VHI	Higher voltage value, corresponding to the logical '1' state that will be reached if the convertor is applied on a node with infinite input resistance	5	V

Table 10-8. .D2A Global Parameters

Name	Description	Default	Units
VLO	Lower voltage value, corresponding to the logical ‘0’ state that will be reached if the convertor is applied on a node with infinite input resistance	0	V
VHIREF	Value of this node is used to set the higher voltage reference value. This corresponds to the logical state ‘1’that will be reached if the boundary element is applied on a node with infinite input resistance.	-	-
VLOREF	Value of this node is used to set the lower voltage reference value. This corresponds to the logical state ‘0’ that will be reached if the boundary element is applied on a node with infinite input resistance.	-	-
TRISE	Rise time for the current source. Commutation time ‘0’ ⇒ ‘1’ value	2ns	s
TFALL	Fall time for the current source. Commutation time ‘1’ ⇒ ‘0’ value	2ns	s
RZ	Effective resistance when input is ‘Z’.	∞	Ω
RRISE	Impedance for a strong logical ‘1’ state. The conductance is then `G=1/RRISE` and the current `I=VHI/RRISE`	1	Ω
RFALL	Impedance for a strong logical ‘0’ state. The conductance is then `G=1/RFALL` and the current `I=VLO/RFALL`	1	Ω
LOWCAP	Capacitance for a logical ‘0’ state (‘L’). This capacitance models the output capacitance of the digital gate	0	F
HIGHCAP	Capacitance for a logical ‘1’ state (‘H’). This capacitance models the output capacitance of the digital gate	0	F
ZCAP	Capacitance for a logical ‘Z’ state. This capacitance models the output capacitance of the digital gate	0	F
XEVAL= PREVIOUS	Forces the corresponding ‘X’ state analog value to the previous analog value before the state changed to ‘X’	-	-
MODE= STD_LOGIC			
WEAHIGHRES	Impedance for a weak (or resistive) logical ‘1’ state (‘H’). The conductance is then `G=1/WEAHIGHRES` and the current `I=VHI/WEAHIGHRES`	1	Ω
WEALOWRES	Impedance for the weak (or resistive) logical ‘0’ state (‘L’). The conductance is then `G=1/WEALOWRES` and the current is `I=VLO/WEALOWRES`	1	Ω

Table 10-8. .D2A Global Parameters

Name	Description	Default	Units
MODE= REAL|INTEGER			
TRISE	Defines the rise time. Commutation time ‘0’ ⇒ ‘1’ value. Ignored if TCOM is specified	2ns	s
TFALL	Defines the fall time. Commutation time ‘1’ ⇒ ‘0’ value. Ignored if TCOM is specified	2ns	s
TCOM	Defines the commutation time. Eldo switches from the previous value to the new value with a slope of time `TCOM`. If specified then TRISE or TFALL parameters are ignored	2ns	s

Parameters

- **`SIM`**`=simulator`

  The parameter **`SIM`** can take the value of the digital simulator’s name: **`HDLA`** or **`VERILOG`**. The parameter **`SIM`** is mandatory only for HDL-A.

- `eldo_node_name`

  Name of the node in the analog netlist.

- `digital_node_name`

  Name of the node in the digital description. This parameter is optional. For HDL-A, the syntax is `Yxx:position` where `position` is the index of the port.

- **`MOD`**`=model_name`

  Name of the model used for the convertor. If `model_name` is specified, there must be a corresponding **`.MODEL`** command:

  ```
  .MODEL model_name D2A|DTOA parameters_list
  ```

- `parameters_list`

  List of available parameters as described below.

Note

For `eldo_node_name` connected to a PORT of an HDL-A model, `parameters_list` is ignored. Default values can depend on simulators (HDL-A, Verilog, and so on).

Parameters for D2A converters (.D2A)

For **`.D2A`** the default **`MODE`** is `X01` for all simulators.

Parameters for MODE=BIT |X01| MVL4 | STD_VSRC

These modes emulate a voltage source for Eldo. Additional parameters:

- **VHI|VOLTHIGH**=value

  Higher voltage value, corresponding to the logical '1' state. Default is 5V.

- **VLO|VOLTLOW**=value

  Lower voltage value, corresponding to the logical '0' state. Default is 0V.

- **VHIREF**=node

  Value of this node is used to set the higher voltage reference, corresponding to the logical '1' state. If specified the value of **VHI|VOLTHIGH** will be ignored.

- **VLOREF**=node

  Value of this node is used to set the lower voltage reference, corresponding to the logical '0' state. If specified the value of **VLO|VOLTLOW** will be ignored.

- **TRISE|TIMERISE**=value

  Defines the rise time for the voltage source, going from VLO to VHI value. Unit is seconds. Default value is 2 ns.

- **TFALL|TIMEFALL**=value

  Defines the fall time for the voltage source going from VHI to VLO value. Unit is seconds. Default value is 2 ns.

Note

For MODE=X01, the logical 'X' state is ignored, that is, the analog value remains the same. For MODE=MVL4, the logical 'X' and 'Z' states are ignored, that is, the analog value remains the same.

Parameters for MODE=X01Z | STD_LOGIC

When the mode of the converter is X01Z|STD_LOGIC|MVL9 these modes emulate an IRC (or IGC) convertor in Eldo. The convertor consists of a current source in parallel with a conductance in parallel with a capacitor with everything connected between the current node and the ground node. Several IRC convertors can be placed on one node and depending on the strength of the signals applied on each of them, the conflicts can be resolved with one of the modes. The values of this IGC vary depending on the state of the D2A convertor, these different values are computed according to the values specified in the D2A statement. If no values are specified, then the default values are used.

Additional parameters for these modes:

- **VHI|VOLTHIGH**=value

  Higher voltage value, corresponding to the logical '1' state that will be reached if the convertor is applied on a node with infinite input resistance. Default value: 5V.

- **VLO|VOLTLOW**=value

  Lower voltage value, corresponding to the logical '0' state that will be reached if the convertor is applied on a node with infinite input resistance. Default value: 0 V.

- **VHIREF**=node

  Value of this node is used to set the higher voltage reference, corresponding to the logical '1' state. If specified the value of **VHI|VOLTHIGH** will be ignored.

- **VLOREF**=node

  Value of this node is used to set the lower voltage reference, corresponding to the logical '0' state. If specified the value of **VLO|VOLTLOW** will be ignored.

- **TRISE|TIMERISE**=value

  Rise time for the current source. Commutation time '0' ⇒ '1' value. Unit is seconds. Default value: 2 ns.

- **TFALL|TIMEFALL**=value

  Fall time for the current source. Commutation time '1' ⇒ '0' value. Unit is seconds. Default value: 2 ns.

- **RZ|HIGHIMPRES**=value

  High impedance to model a high impedance state ('Z'). In this case, the conductance (G=1/RZ) will be small, and the current I=0. Default is 1Ω.

- **RRISE|STRHIGHRES**=value

  Impedance for a strong logical '1' state. The conductance is then G=1/RRISE and the current I=VHI/RRISE. This resistance is usually small. Default is 1Ω.

- **RFALL|STRLOWRES**=value

  Impedance for a strong logical '0' state. The conductance is then G=1/RFALL and the current I=VLO/RFALL. This resistance is usually small. Default is 1Ω.

- **LOWCAP**=value

  Capacitance for a logical '0' state ('L'). This capacitance models the output capacitance of the digital gate. Default is 0 F.

- **HIGHCAP**=value

  Capacitance for a logical '1' state ('H'). This capacitance models the output capacitance of the digital gate. Default is 0 F.

- **ZCAP**=value

  Capacitance for a logical 'Z' state. This capacitance models the output capacitance of the digital gate. Default is 0 F.

- **XEVAL**=**PREVIOUS**

  When this is specified, the corresponding 'X' state analog value will be the previous analog value before the state changed to 'X', that is, if the digital state changed from 'L' => 'X' then the analog value will stay at **VLO**. The default value is computed from the impedances set in **.D2A** or **.MODEL** statements for the 'X' states.

Parameters for MODE=STD_LOGIC

- **WEAHIGHRES**=value

  Impedance for a weak (or resistive) logical ‘1’ state (‘H’). The conductance is then G=1/WEAHIGHRES and the current I=VHI/WEAHIGHRES. This resistance is of course larger than RFALL and RRISE. Default is 1Ω.

- **WEALOWRES**=value

  Impedance for the weak (or resistive) logical ‘0’ state (‘L’). The conductance is then G=1/WEALOWRES and the current is I=VHI/WEALOWRES. This resistance must be larger than RFALL and RRISE. Default is 1Ω.

Logical states

‘X’ State

For the logical ‘X’ state which is also referred to as ‘strong X’, the corresponding conductance (GX), capacitance (CX), transition time (TX) and current source (IX) values are calculated as shown below:

$$GX = \frac{RRISE^{-1} + RFALL^{-1}}{2}$$

$$CX = \frac{LOWCAP + HIGHCAP}{2}$$

$$TX = \frac{TRISE + TFALL}{2}$$

$$IX = \frac{(VHI + VLO)}{2} \times GX$$

‘W’ State

For the logical ‘W’ state which is also referred to as ‘strong W’ the corresponding conductance (GX), capacitance (CX), transition time (TX) and current source (IX) values are calculated as shown below:

$$GX = \frac{WEAHIGHRES^{-1} + WEALOWRES^{-1}}{2}$$

$$CX = \frac{LOWCAP + HIGHCAP}{2}$$

$$TX = \frac{TRISE + TFALL}{2}$$

$$IX = \frac{(VHI + VLO)}{2} \times GX$$

Note
The capacitance (CX) and transition time (TX) are the same for both 'X' and 'W' states.

Parameters for MODE=REAL|INTEGER

Eldo will read from a DIGITAL simulator the real value instead of a logical value. Additional parameters:

- **TRISE|TIMERISE**=value

  Defines the rise time. Commutation time '0' ⇒ '1' value. Unit is seconds. Default value: 2 ns. Ignored if **TCOM** is specified.

- **TFALL|TIMEFALL**=value

  Defines the fall time. Commutation time '1' ⇒ '0' value. Unit is seconds. Default value: 2 ns. Ignored if **TCOM** is specified.

- **TCOM**=value

  Defines the commutation time. Eldo switches from the previous value to the new value with a slope of time TCOM. Default is 2 ns. For backward compatibility, when **TCOM** is specified, the parameters **TRISE** or **TFALL** will be ignored and a warning generated.

Example: CMOS model

```
.D2A SIM=HDLA D2A_STD_LOGIC MOD=D2A_B_STD_LOGIC
.MODEL D2A_B_STD_LOGIC D2A MODE=STD_LOGIC
+ VLO=0.0 VHI=5.0 TRISE=3N TFALL=2N
+ RZ=1.0E+12 RRISE=1000 RFALL=1000
+ WEAHIGHRES=10K WEALOWRES=10K
+ LOWCAP=0.1p HIGHCAP=0.1p ZCAP=0.1p
```

V source converter for STD_LOGIC

A type of D2A converter, FAST_STD_LOGIC, is implemented. This FAST_STD_LOGIC converter can improve simulation speed, especially when used in ADMS and ADiT. The FAST_STD_LOGIC boundary element can be applied to A2D boundaries, in this case the boundary element is equivalent to a boundary element of type STD_LOGIC.

The STD_LOGIC D2A models are very flexible; they can handle high-impedance states and signals of different strengths. However, simulation can be longer compared with a D2A acting as a voltage source, as is the case for the BIT model. The FAST_STD_LOGIC converter overcomes this limitation.

When interfacing a Verilog or VHDL Std_logic signal with analog nets, it is possible to manage conflicts of a 'Z' state with a V source built-in boundary element of type FAST_STD_LOGIC which can handle only 0, 1, X and Z states.

The V source built-in boundary element can be selected by setting the **MODE** of the converter to **FAST_STD_LOGIC** or **STD_VSRC** (they are synonymous). It has the same set of parameters as **MODE**=BIT or **MODE**=X01 Eldo converters.

Eldo checks whether there are one or more D2A boundary elements present, which are not in a high-impedance ('Z') state. If there are, Eldo will take into account all the D2A signals on the node and will compute the resulting voltage value. This value will be imposed on the analog part as a voltage source would. This means that the voltage on that node is not an *unknown* to the system of equations, and this can make simulation faster. If, however, there are only high-impedance states on the D2A node, Eldo will solve for that node.

Option DEFD2A

It is normally necessary to specify an explicit D2A between ANALOG nodes and HDL-A ports; however, with the following statement specified:

```
.OPTION DEFD2A=model_name
```

Eldo will automatically insert an implicit D2A convertor when needed. The following must be defined in the netlist:

```
.MODEL model_name D2A parameters
```

Furthermore, it is often necessary to specify **MODE**=**BIT** in the **.MODEL** model_name command. This is because the default for **MODE** is X01, and very often HDL-A PORTS are of type **BIT**, hence a resulting error mismatch in convertor type would be printed if **MODE** is not correctly specified.

Example

```
.MODEL model_name D2A mode = BIT
.OPTION DEFD2A = model_name
```

Additionally, these default models must be found in the netlist itself, and cannot be specified via **.LIB** or **.ADDLIB** specifications.

> **Note**
> With the option **DEFD2A**, for any ANALOG node connected directly to an HDL-A **OUT** or **INOUT** port, an implicit D2A will be automatically inserted.

Option D2DMVL9BIT

Enables direct connection between HDL-A ports of type BIT with mixed-mode nodes connected to Eldo via convertors of type MVL9 (or std_logic).

The **BIT** type is a subset of std_logic so there is no conversion: '0' -> '0' and '1' -> '1'.

.DATA

Parameter Sweep

```
.DATA dataname parameter_list
[+] val_list1
[+] val_list2
[+] ...
[.ENDDATA]

.DATA dataname MER[GE]|LAM[INATED]
[+] file=filename1 param=column ... param=column
[+] file=filename2 param=column ...
[+] ...
[+] [out=outfile]
[.ENDDATA]
```

Two forms of **.DATA** statement definition are allowed. One is the inline form, and the other is where the statements are defined using external files. The data from multiple external files can be processed sequentially or in parallel.

Parameters

- `dataname`

  The name assigned to the **.DATA** statement, this name could be used in **.TRAN**, **.DC** or **.AC** commands using the format: **SWEEP DATA**=dataname

- `parameter_list`

  List of n parameter names.

- `val_list`

  Set of n double values corresponding to the values of the `parameter_list`.

- **MER**[**GE**]|**LAM**[**INATED**]

  Forces Eldo to process the data statements from multiple external files sequentially (**MER**[**GE**]) or in parallel (**LAM**[**INATED**]).

- **file**=filename

  Definition of **.DATA** statements using external files. The data from multiple files can be processed sequentially or in parallel.

- **param**=column

  Defines the column number from which the parameter values are taken.

- **out**=outfile

  Specifies the output file in which the resulting **.DATA** statements can be saved.

Notes

1. The "+" continuation characters are not required for the list of values when using this command. This differs from other Eldo commands where the "+" continuation characters are required.

2. Implicit **.DATA** creation is also possible:

   ```
   .TRAN 1n 10n SWEEP p2 3.0 4.0 1.0
   ```

   Eldo will make two TRAN runs: one with `p2` set to 3.0, another with `p2` set to 4.0.

3. **.ENDDATA** is optional.

4. Eldo will generate an ASCII output file with extension *.mt#* (for transient), *.ma#* (for AC) and *.md#* (for a DC sweep). The # character stands for the index of the file and a new file is created for each new **.TEMP** or **.ALTER** command.

5. It is possible to have the input signal being read from **.DATA** commands. An example is shown below:

   ```
   .DATA srcname
   + time pv1
   + 0 0
   + 5n 0
   + 20n 5
   .ENDDATA
   V1 1 0 PWL (time,pv1)
   .TRAN data=srcname
   ```

   Here, the signal on **V1** will be read from the **.DATA** command. This works only if the following two conditions apply:

   a. The **.TRAN** command is defined via a **.DATA** command. Time will be assumed to be the first parameter name in the **.DATA** command, and simulation will proceed until the last value specified in the **.DATA** command.

   b. Both parameters which appear in the **PWL** must be in the same **.DATA** command.

6. **.MPRUN** can be used to take advantage of multi-processor machines for the **.DATA** command. Please see .MPRUN for further information.

7. SOA limits can be defined using **.DATA** statements. Please see .SETSOA for further information.

8. A wave defined with a a **.DATA** statement can be plotted with the **DATA**(`dataname, parameter`) plot specification.

Examples

```
.TRAN 1n 10n SWEEP DATA=datatran
.DATA datatran p1 p2 p3
1.0 3.0 4.0
2.0 5.0 7.0
.ENDDATA
```

Eldo will make two TRAN runs: the first with `p1`=1.0, `p2`=3.0, and `p3`=4.0, and the second with `p1`=2.0, `p2`=5.0, and `p3`=7.0. Note the “+” continuation characters are not required.

Here follow some examples of specifying multiple external files to be processed either sequentially or in parallel:

```
.data example1 MERGE
+ file=data1.inc col1=3 col2=2
+ file=data2.inc colA=1 colB=3
+ out=example1.inc
.enddata
```

After the first file name, the names of the columns to be created and the index where the data comes from are specified. The resulting **`.DATA`** is created using the merge of the two files:

```
 1   2   3
 4   5   6
 7   8   9
10  20  30
40  50  60
70  80  90
```

The specification col1=3 col2=2 and colA=1 colB=3 literally means: the first column is named col1 and its values come from the third column of merged files; the second column is named col2 and its values come from the second column of merged files. The resulting **`.DATA`** will be:

```
.DATA example1
+ col1  col2  colA  colB
+  3     2     1     3
+  6     5     4     6
+  9     8     7     9
+ 30    20    10    30
+ 60    50    40    60
+ 90    80    70    90
.enddata
```

The following example shows the **`LAMINATED`** specification for processing in parallel:

```
.data example2 LAM
+ file=data1.inc col1=3 col2=2
+ file=data2.inc colA=1 colB=3
+ out=example2.inc
.enddata
```

As before, the names of the columns to be created and the index where the data comes from are specified. However, unlike the merge form, values come from the original files so in this case you will notice that the resulting **`.DATA`** will be different as follows:

```
.DATA example2
+ col1  col2  colA  colB
+  3     2     10    30
 + 6     5     40    60
 + 9     8     70    90
.enddata
```

.DC

DC Analysis

Single Analysis

```
.DC
```

Component Analysis

```
.DC CNAM [L|W] [TYPE nb] START STOP INCR [SWEEP DATA=dataname] [MONTE=val]
.DC CNAM [L|W] [TYPE nb] START STOP INCR
+ [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC CNAM [L|W] [TYPE nb] START STOP INCR
+ [SWEEP parameter_name start stop incr] [MONTE=val]
```

Voltage or Current Source Analysis

```
.DC SNAM [TYPE nb] START STOP INCR [SNAM2 START2 STOP2 INCR2]
+ [SWEEP DATA=dataname] [MONTE=val]
.DC SNAM [TYPE nb] START STOP INCR [SNAM2 START2 STOP2 INCR2]
+ [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC SNAM [TYPE nb] START STOP INCR [SNAM2 START2 STOP2 INCR2]
+ [SWEEP parameter_name start stop] incr [MONTE=val]
```

Temperature Analysis

```
.DC TEMP START STOP INCR [SWEEP DATA=dataname] [MONTE=val]
.DC TEMP START STOP INCR [SWEEP parameter_name TYPE nb start stop]
+ [MONTE=val]
.DC TEMP START STOP INCR [SWEEP parameter_name start stop incr]
+ [MONTE=val]
```

Parameter Analysis

```
.DC PARAM PARAM_NAME START STOP INCR [SWEEP DATA=dataname]
+ [MONTE=val]
.DC PARAM PARAM_NAME START STOP INCR
+ [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC PARAM PARAM_NAME START STOP INCR
+ [SWEEP parameter_name start stop incr] [MONTE=val]
.DC PARAM PARAM_NAME [TYPE nb] START STOP INCR
```

Data-Driven Analysis

```
.DC DATA=dataname [SWEEP DATA=dataname] [MONTE=val]
.DC DATA=dataname [SWEEP parameter_name TYPE nb start stop] [MONTE=val]
.DC DATA=dataname [SWEEP parameter_name start stop incr] [MONTE=val]
```

The **.DC** command activates a DC analysis, and is used to determine the quiescent state or operating point of the circuit. The operating point of the circuit is computed with capacitances opened and inductances short-circuited. A DC analysis may be requested to determine the stable initial condition of an analog circuit, prior to a transient or AC analysis. Three levels of nesting are allowed.

There are six different types of DC analysis available as shown below:

1. **DC** with no further parameters results in a single analysis of the circuits' quiescent state.
2. **DC** followed by a component name results in a variation of the element size or value. The variation sweeps from the value START to the value STOP with the incremental step INCR. The quiescent state is calculated for each incremental step.
3. **DC** followed by a voltage or current source, results in a voltage or current sweep of the specified source from START to STOP in increments INCR. The quiescent state is calculated for each incremental step. A second source SNAM2 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.
4. **DC** followed by the **TEMP** keyword results in a variation of temperature. The temperature sweeps from the value of START to the value of STOP with the incremental step INCR. The quiescent state is calculated for each incremental step.
5. **DC** followed by the **PARAM** keyword results in a variation of the value of a globally declared parameter PARAM_NAME. The variation sweeps from the value of START to the value of STOP with the incremental step INCR. The quiescent state is calculated for each incremental step.
6. **DC** followed by the parameter **DATA**=dataname results in a sweep of the values pre-defined in the **.DATA** command. The quiescent state is calculated for each incremental step.

Multi-threading can be activated for a single DC simulation, Eldo will share computer resources on a multi-processor machine. Alternatives are:

- command line flag **-mthread** (see "-mthread" on page 50) at Eldo invocation, or option **MTHREAD** in the netlist. Eldo will make use of all the possible CPUs on the machine.
- command line flag **-usethread #** (see "-usethread val" on page 56) at Eldo invocation, or option **USETHREAD**=val. This forces Eldo to use at maximum the specified (#) number of CPU. The number specified can exceed the number of CPUs available, but this is not recommended, even though Unix will allow it.

Statistics, generated at the end of simulation, show how many CPUs have been used for the current simulation.

Parameters

- CNAM

  Name of component on which geometrical or value variations are performed. Acceptable components are inductors, resistors, MOS transistors and controlled sources (VCVS, CCVS, CCCS, VCCS). For the control sources, only the linear case is supported and only the gain of these sources may be swept.

- START

  Start value of the component CNAM, voltage, temperature or current sweep. This can be specified as a parameter or as an expression.

- STOP

  Stop value of the component CNAM, voltage, temperature or current sweep. This can be specified as a parameter or as an expression.

- INCR

  Increment of the component, voltage, temperature, or current sweep. This can be specified as a parameter or as an expression.

- SNAM

  Name of the voltage or current source which performs the DC sweep.

- **TEMP**

  Keyword indicating that the temperature is to be varied.

- **PARAM**

  Keyword indicating that a parameter is to be varied.

- **DATA**=dataname

  Used in conjunction with the **.DATA** command. The dataname parameter should be specified using the **.DATA** command. Please refer to “.DATA” on page 585 for more information.

- PARAM_NAME
- Name of the globally declared parameter to be varied. Both primitive and non-primitive parameters are allowed.
- **L**, **W**

  Keywords which determine if the length **L** or the width **W** of the MOS component CNAM is to be varied.

- TYPE

  Type name of the first level of variation for DC component analysis and voltage/current source analysis. Can be one of the following:

  DEC

  Keyword to select logarithmic variation.

  OCT

  Keyword to select octave variation.

  LIN

  Keyword to select linear variation.

POI

Keyword to select a list of frequency points. **POI** is the same as **LIST** except that **POI** expects the number of points nb to be specified as it's first argument.

- SNAM2

  Name of the secondary voltage or current source for DC sweep.

- START2

  Start value of the secondary voltage or current sweep. This can be specified as a parameter or as an expression.

- STOP2

  Stop value of the secondary voltage or current sweep. This can be specified as a parameter or as an expression.

- INCR2

  Increment of the secondary voltage or current sweep. This can be specified as a parameter or as an expression.

- **MONTE**=val

  Monte Carlo analysis. Equivalent to **.MC** val. The syntax allows a different **MONTE** value for each run. However, the actual implementation in Eldo does not account for that: it is the last **MONTE** value to be specified in the netlist which will be taken into account.

Sweep Parameters

This section contains **SWEEP** related parameters that are previously unspecified in the Parameters section.

- **SWEEP**

  Specifies that a sweep should be performed on a parameter or device name.

- parameter_name

  Name of the parameter or device name to sweep.

- TYPE

  Can be one of the following:

  DEC

  Keyword to select logarithmic variation.

  OCT

  Keyword to select octave variation.

  LIN

  Keyword to select linear variation.

POI

Keyword to select a list of frequency points. **POI** is the same as **LIST** except that **POI** expects the number of points nb to be specified as it's first argument.

INCR

Increment of the parameter or device name to sweep. (**INCR** can only be used when the **SWEEP** keyword is preceding it. When **INCR** is specified, nb is the incrementing value.)

- nb

  Number of points required.

- start

  Start value of the parameter or device.

- stop

  Stop value of the parameter or device.

- incr

  Increment of the parameter or device name to sweep.

Note

When **INCR** is specified as the TYPE parameter, the value which directly follows (nb) is the incrementing value. If **INCR** is not specified, the incrementing value (incr) must be placed after the start and stop values.

Examples

```
vin 1 0 10
.dc vin 0 5 0.2
```

Specifies a DC analysis with voltage sweep of the voltage source **vin** from 0 to 5 V with an increment of 0.2 V.

```
r7 3 4 100k
.dc r7 10k 100k 10k
```

Specifies a DC analysis with resistor value variation of **r7** from 10 kΩ to 100 kΩ with increments of 10 kΩ.

```
r1 1 2 p1
.param p1=1k
.dc param p1 1k 10k 1k
```

Specifies a DC analysis with resistor parameter p1 variation from 1 kΩ to 10 kΩ with an increment of 1 kΩ.

```
VIN 1 0 10
.param p1=3
.param pend=10
.DC VIN p1 pend 0.1
```

Specifies a DC analysis with voltage sweep of the voltage source `vin` from 3 to 10V with an increment of 0.1V. This example shows how the **`.DC`** command accepts parameters as arguments.

```
.dc v1 dec 10 2 12 sweep r2 INCR 10 1k 100k
.dc v1 dec 10 2 12 sweep r2 1k 100k 10
```

The two lines above are equivalent. Either can be used to specify a DC analysis at each of the values for `v1` starting at 2V and stopping at 12V with 10 analysis points per decade. The sweep specification will force Eldo to carry out a DC analysis on each value of `r2` starting at 1kΩ and stopping at 100kΩ with an incrementing value of 10Ω.

```
.dc E1 3 5 0.5
```

In the sweep of gain for linear controlled sources example above, the gain of the VCVS E1 will be swept from 3 to 5 in increments of 0.5.

The next example shows that the DC sweep can be nested up to three levels deep. For each value of the third level variable, the second level variable is swept through its specified range. For each value of the second level variable, the first level variable is swept through its specified range. To show the order of sweepings, the below shows three independent voltage sources swept with the .DC statement.

```
v1    1    0 0v
v2    2    0 0v
v3    3    0 0v
.dc v1 1 2 0.5   v2 2 3 0.5     v3 3 4 0.5
.print dc v(1) v(2) v(3)
```

The .PRINT statement results in the following output in the *.chi* file:

```
 V1          V(1)         V(2)         V(3)

1.0000E+00   1.0000E+00   2.0000E+00   3.0000E+00
1.5000E+00   1.5000E+00   2.0000E+00   3.0000E+00
2.0000E+00   2.0000E+00   2.0000E+00   3.0000E+00
1.0000E+00   1.0000E+00   2.5000E+00   3.0000E+00
1.5000E+00   1.5000E+00   2.5000E+00   3.0000E+00
2.0000E+00   2.0000E+00   2.5000E+00   3.0000E+00
1.0000E+00   1.0000E+00   3.0000E+00   3.0000E+00
1.5000E+00   1.5000E+00   3.0000E+00   3.0000E+00
2.0000E+00   2.0000E+00   3.0000E+00   3.0000E+00
1.0000E+00   1.0000E+00   2.0000E+00   3.5000E+00
1.5000E+00   1.5000E+00   2.0000E+00   3.5000E+00
2.0000E+00   2.0000E+00   2.0000E+00   3.5000E+00
1.0000E+00   1.0000E+00   2.5000E+00   3.5000E+00
1.5000E+00   1.5000E+00   2.5000E+00   3.5000E+00
2.0000E+00   2.0000E+00   2.5000E+00   3.5000E+00
1.0000E+00   1.0000E+00   3.0000E+00   3.5000E+00
1.5000E+00   1.5000E+00   3.0000E+00   3.5000E+00
2.0000E+00   2.0000E+00   3.0000E+00   3.5000E+00
1.0000E+00   1.0000E+00   2.0000E+00   4.0000E+00
1.5000E+00   1.5000E+00   2.0000E+00   4.0000E+00
2.0000E+00   2.0000E+00   2.0000E+00   4.0000E+00
```

```
1.0000E+00   1.0000E+00   2.5000E+00   4.0000E+00
1.5000E+00   1.5000E+00   2.5000E+00   4.0000E+00
2.0000E+00   2.0000E+00   2.5000E+00   4.0000E+00
1.0000E+00   1.0000E+00   3.0000E+00   4.0000E+00
1.5000E+00   1.5000E+00   3.0000E+00   4.0000E+00
2.0000E+00   2.0000E+00   3.0000E+00   4.0000E+00
```

The following examples show how the keywords **DEC**, **OCT**, **LIN**, and **POI** can be specified as the type name of the first level of variation for DC component analysis and voltage/current source analysis.

```
.dc r1 DEC 8 2k 9k
.dc r1 OCT 8 2k 9k
.dc r1 LIN 100 2k 9k
.dc r1 POI 8 2k 3k 4k 5k 6k 7k 8k 9k
```

.DCHIZ

DC Analysis High Impedance Detection

```
.DCHIZ R=value
```

Detects high impedance values for DC analysis. All impedances higher than a user-specified value are listed. The analysis is performed just after DC analysis.

Parameters

- **R**=value

  Specifies the high impedance value. All impedances above this value will be listed.

Example

```
V1 1 0 1.9
R1 1 2 101
R1p 2 2p 101
MC1 2p G 3 3 NMOS W=0.1u L=0.1U

R2 3 0 102
VG GP 0  1.8v
R12 GP G 100m
.param limit_value=10.1
.TRAN 10ns 1us
.PLOT TRAN V(*)
.DCHIZ R=limit_value
.MODEL NMOS NMOS LEVEL=1
```

All nodes which have an equivalent impedance higher than limit_value will be displayed and written to the *.chi* file, for example:

```
****     HIGH IMPEDANCE DETECTION
HIGH IMPEDANCE on node "2" Impedance = 1.010000e+02
HIGH IMPEDANCE on node "2P" Impedance = 2.020000e+02
HIGH IMPEDANCE on node "3" Impedance = 1.016275e+02
```

.DCMISMATCH

DC Mismatch Analysis

```
.DCMISMATCH [FILE=filename] [output] [DCALL[=0|1]]
+ [SORT_REL=value] [SORT_ABS=value] [SORT_NBMAX=value] [NSIGMA=value]
```

This command is a special form of sensitivity analysis based on statistical deviation properties of device model parameters. It computes the sensitivity of node voltages and of currents through voltage sources. All devices using a device model with statistical deviations potentially contribute to the output deviation. The aim of the **.DCMISMATCH** analysis is to quantify these contributions and to identify the biggest contributors to the total output deviation. It has some degree of similarity with the worst-case analysis (**.WCASE**). However, it will usually provide (much) less pessimistic results than worst-case analysis.

.DCMISMATCH will consider exclusively the **DEV** or **DEVX** variations specified for device model parameters (in **.MODEL** statements) and for global parameters in the *.cir* file (**.PARAM** statements specified at the top-level, outside any subcircuit definition). The **LOT** specifications are not taken into account.

Mismatch models try to model the inherent differences which exist between two identically drawn transistors. Each transistor behavior departs (statistically) from the nominal behavior. The deviation depends strongly on the geometry (W, L) of the transistor. Small transistors tend to be more random than large ones.

.DCMISMATCH produces an ASCII report in the main output (*.chi*) file, which lists the output deviations due to the deviations of the sensitive devices and parameters. Specifically, deviation is applied independently to the sensitive devices, and the resulting deviation of the output is computed from the RMS sum of the resulting deviations. The mismatch report can be generated in a separate file if requested.

The `SORT_` parameters are used to limit the amount of output information.

The keyword, **STATISTICAL**= 0|1, can be specified on X instances, device declarations, or on **.SUBCKT** definitions, to specify whether any statistical variation due to DC mismatch analysis can be applied to the specified entities. If **STATISTICAL** is 0, the selected devices will keep their nominal values. If **STATISTICAL** is 1, the selected devices have statistical variation applied. The global default can be specified via option **STATISTICAL**= 0|1. Default is 1.

Note

Multiple sensitivity and statistical analyses cannot be used simultaneously. Specifically, **.DCMISMATCH**, **.MC**, and **.WCASE** are exclusive. Only one of them can be specified in a netlist.

DC mismatch information can also be extracted with the **.EXTRACT DCM** function, which returns the result for a specific `label_name`. It extracts the value which is dumped in the *.chi* file.

There are two methodologies when using the **.DCMISMATCH** command:

- **.DCMISMATCH** with no output specified, the analysis is performed automatically for all DC extracts (associated **.EXTRACT** statements required) to obtain the result.
- **.DCMISMATCH** with V or I output specified, in this case, **.EXTRACT DCM** statements should not be specified (they are not taken into account).

Extractions using general purpose functions, such as yval, xycond, cannot be used for **.DCMISMATCH** analysis. Only extractions using a combination of I(device) or V(node) quantities are allowed.

Parameters

- **FILE**=filename

  Optional. Specifies a separate output file to be created containing mismatch results. When specified, the mismatch report will be written to both the *.chi* file and the file specified by this parameter.

- output

  Optional. The output can be **V**(net_name1[,net_name2]), **I**(Vsrc), **EXTRACT**(label), or **MEAS**(label). If no output is specified, the analysis is performed for all DC extracts.

- DCALL[=0|1]

  Optional. If set to 1, then the **.DCMISMATCH** analysis will be performed on all points of the DCSWEEP. If set to 0, then it will be performed only on the first one.

- SORT_REL

  Only devices with a contribution to the output greater than the value: SORT_REL×<MAX_variation_on_output> will be listed. The default value of SORT_REL is 0.001, which means that all devices contributing to more than 1/1000 of the maximum variation will be listed.

- SORT_ABS

  Used to specify the absolute threshold, below which contributors will not be listed. Default value is 0.

- SORT_NBMAX

  Allows the user to limit the list of contributors to a certain value. By default, all contributors are listed.

Note

The sorting parameters are additive, that is, their respective effects are cumulative. For example, to create a report with only devices contributing to 10% or more of the maximum output deviation and have 15 contributors at most you would specify:
SORT_REL=0.1
SORT_NBMAX=15.

- `NSIGMA`

  Parameter used to change the default sigma value used for Gaussian distributions in Eldo statistical analysis.

The total variation on the output is computed from the 4-sigma variation for Gaussian distribution, or from the max variation for other distributions (UNIFORM distribution or distribution defined with a **`.DISTRIB`** command). 4- sigma is the default used for Gaussian distributions in Eldo statistical analyses. This can be changed with the `NSIGMA` value of the **`.DCMISMATCH`** command, and if not specified, from the global option **`SIGTAIL=val`**, which defaults to 4.

Note

With gaussian distributions, 4-sigma deviations may cause unrealistic situations if the standard deviation is too large, so this should be handled with care. For example specifying `DEV`=10% for a 0.35V threshold voltage with a Gaussian distribution will most probably cause a problem when a -4 sigma variation is applied.

Examples

These first two examples below show how **`.DCMISMATCH`** will consider the **`DEV`** variations specified for device model parameters and for global parameters:

```
.model nch nmos level=53 vt0=0.45 dev=1%...
```

All transistors using the **`nch`** model will be considered.

```
.param vdd=1.2V dev=5%
```

The **`vdd`** parameter will be considered.

```
.extract dc label=VS V(s)
.extract dc label=dcmm_vs dcm(VS)
.dcmismatch nsigma=1 sort_nbmax=5
```

Here Eldo will return the variation of all the valid DC extracts, namely the EXTRACT labelled VS (the second extract dcmm_vs is just for output, it will not be seen by the mismatch). The above is equivalent to the following:

```
.extract dc label=dcmm_vs dcm(VS)
.dcmismatch V(s) nsigma=1 sort_nbmax=5
```

or:

```
.extract dc label=VS V(s)
.extract dc label=dcmm_vs dcm(VS)
.dcmismatch meas(VS) nsigma=1 sort_nbmax=5
```

Below is a typical DC mismatch analysis output showing N-sigma deviation of the outputs, and a table of sorted contributors. In this example, XM2.M1 is the main contributor, and the deviation of the U0 mobility parameter of its associated .model (XM2.MOD2) is the main contributor to the XM2.M1 contribution.

```
#.DCMISMATCH  V(S) SORT_REL = 1.000000e-03 NSIGMA = 1.000000e+00

Analysis results:

Output DC value                     :     1.7892U     Volt
Total output deviation              : +/- 14.9166M    Volt
Output deviation due to the
contributors in the report table    : +/- 14.9166M    Volt

Report table
----------------------------------------

 Output deviation       Contributor      Parameter
 7.9308M                 XM2.M1
             5.1779M                        M(XM2.MOD2,U0)
             6.0051M                        M(XM2.MOD2,VTH0)
             163.0689U                      M(XM2.MOD2,TOX)
 7.9294M                 XM1.M1
             5.1769M                        M(XM1.MOD2,U0)
             6.0040M                        M(XM1.MOD2,VTH0)
             163.0386U                      M(XM1.MOD2,TOX)
 6.1906M                 XM7.M1
             2.5507M                        M(XM7.MOD1,U0)
             5.6326M                        M(XM7.MOD1,VTH0)
             302.4733U                      M(XM7.MOD1,TOX)
 6.1906M                 XM6.M1
             2.5507M                        M(XM6.MOD1,U0)
             5.6326M                        M(XM6.MOD1,VTH0)
             302.4703U                      M(XM6.MOD1,TOX)
...
```

.DEFAULT

Set Default Conditions

```
.DE[FAULT] TYPE VALUE
.DE[FAULT] TYPE {KEYWORD [VALUE]}
```

This command resets the default values for elements, device initial conditions and model parameters.

The first syntax shown above is the general form for resistors, capacitors, and inductors. The second syntax is the general form for other types.

You may specify only one element or model type per **.DEFAULT** statement. However, in your circuit netlist file you may include as many **.DEFAULT** statements as required to set all required default conditions.

Parameters

- TYPE

  Type of element or model.

- KEYWORD

  Any valid element or model parameter or keyword.

- VALUE

  Optional value given to keywords.

Limitation

Eldo implementation of **.DEFAULT** does not support the Lossy Transmission Line (LDTL) model and IC for active devices are ignored.

Example

.DEFMAC

Macro Definition

```
.DEFMAC MAC_NAME(ARG{, ARG})=EXPRESSION
```

The **.DEFMAC** command is used to define a parameterized macro which may be instantiated (used) in the circuit netlist. The macro may contain a combination of arithmetic expressions or pre-defined Eldo functions. The macro is instantiated using the **.EXTRACT** command and results are listed to the ASCII output (*.chi*) file.

Parameters

- MAC_NAME

  Macro name.

- ARG

  Argument names passed to the macro at instantiation time.

- EXPRESSION

  A combination of arithmetic expressions and pre-defined function calls that may use the arguments passed to the macro.

Note

Argument names cannot contain arithmetic operators.
Only one macro may be defined per **.DEFMAC** statement.
TRISE, TFALL, TPDxx, SLEWRATE, RMS, INTEG, DW_A cannot be used inside **.DEFMAC**.
Only the so called "general purpose extraction language" terms (END, MAX, MIN, START, XAXIS, XYCOND, XUP, XDOWN, YVAL) can be used.
A macro instantiation is made by preceding the macro name with the $ character.
Keywords **XAXIS**, **END** and **START** are recognized by the **.DEFMAC** statement. This simplifies the writing of macros.

Examples

```
.defmac phmag(a, b)=xycond(a, b<0.0)-yval(a, 0.001)
...
.extract $phmag(vp(s), vdb(s))
```

This example defines a macro called phmag with arguments a and b. This macro is then instantiated via the **.extract** command with the arguments a and b being replaced by the phase voltage vp(s) and magnitude vdb(s) on node s respectively. The results are listed in the ASCII output file. The phmag macro uses the pre-defined functions **xycond** and **yval**.

For more information on these functions, refer to ".EXTRACT" on page 637.

```
.defmac sett(out,ratio) =
+ XYCOND(XAXIS,(out > (yval(out,END) * (1 + ratio))) ||
+ (out < (yval(out,END) * ( 1 - ratio))),END,START)
.extract sett(v(s),0.1)
```

Returns the settling time of `v(s)`, that is, the time at which `v(s)` does not vary outside the range 0.9×`final_value`, 1.1×`final_value`, where `final_value` is the value of `v(s)` at the end of the simulation.

```
.defmac sett(x,out,tx,ratio) =
+ XYCOND(x,(out > (yval(out,tx)*(1 + ratio))) ||
+ (out < (yval(out,tx) * ( 1 - ratio))),tx,0)
.extract $sett(xaxis,v(s),50n,0.1)
```

This is for finding the settling time on wave `out`.

.DEFMOD

Model Name Mapping

.DEFMOD alias_model_name actual_model_name

This command can be used to map a model name in a netlist to a model name specified in a **.MODEL** card. This works if the **.DEFMOD** is placed before any use of the string alias_model_name.

This is similar to the .MALIAS command, except the arguments are reversed.

Parameters

- alias_model_name

  Model name given in a netlist.

- actual_model_name

  Model name defined in **.MODEL** card.

Example

```
.model model1 r tc1=2 tc2=1
.defmod modalias model1
r1 1 0 modalias r=1k
```

Model name model1 is aliased to modalias.

.DEFPLOTDIG

Plotting an Analog Signal as a Digital Bus

```
.DEFPLOTDIG [VTH[1]=VAL [VTH2=VAL]]
```

The **.DEFPLOTDIG** command enables the user to plot an analog signal as a digital bus. This command can *only* be used as a precursor to **VDIG** of the **.PLOT** command.

Parameters

- **VTH[1]=**VAL

  If a voltage threshold is specified, the bus of an analog signal is plotted as a bus (hexadecimal format), else all the different signals of the bus are plotted separately in the wave viewer as analog waves. (**VTH** and **VTH1** are synonymous to ensure backwards compatibility.)

- **VTH2=**VAL

  Can be used to plot the indeterminate value, as shown below:

  When only **VTH1** is given:
  If value < **VTH** then logic state 0.
  If value > **VTH** then logic state 1.

  When both **VTH1** and **VTH2** are given:
  If value < **VTH1** then logic state 0.
  If **VTH1** < value < **VTH2** then state X.
  If value > **VTH2** then logic state 1.

Example

```
.DEFPLOTDIG VTH1=2.2 VTH2=2.7
.PLOT TRAN VDIG(n2)
```

When **.DEFPLOTDIG** is used in conjunction with **.PLOT VDIG**, the signal node **n2** is plotted as a digital curve and will only have values of “1” or “0”.

.DEFWAVE

Waveform Definition

```
.DEFWAVE [SWEEP] [ANALYSIS] WAVE_NAME=WAVE_EXPR
```

The **.DEFWAVE** command is used to define a new waveform by relating previously defined waveforms and nodes. The waveform definition may contain a combination of arithmetic expressions or pre-defined functions available in Eldo. The waveform may be instantiated using the **.EXTRACT** command with the results being listed to the ASCII output (*.chi*) file or may be printed or plotted using the **.PLOT** and **.PRINT** commands.

Defwaves defined during an analysis working in complex mode (AC, NOISE, RF, and so on) are always considered complex even if they contain pure real values (in which case the imaginary part is 0.0). In the following example, the phase transformation is applied twice:

```
.defwave ac defwave_phase= vp(4)
.plot ac vp(4) wp(defwave_phase)
```

Parameters

- **SWEEP**

  When a circuit contains some **.EXTRACT** or **.MEAS** commands, and in the case of **.STEP** cards, it applies a wave operator on the curves **EXTRACT**=F(STEP), that is, those waves dumped in the *.ext* file.

- ANALYSIS

  Type of analysis to be used.

- WAVE_NAME

  New waveform name.

- WAVE_EXPR

  Arithmetic expression relating previously defined waveforms and nodes. See “Arithmetic Functions” on page 78 for further details.

  The function types **PWL**, **PWL_CTE** and **PWL_LIN** are also available for **.DEFWAVE**, see Table 10-9. They create piece-wise linear (PWL) waves which can be plotted in the binary output (*.wdb*) file. These functions are used for creating an ideal waveform which can then be displayed in EZwave and matched against actual simulation results. See the example in Figure 10-3 on page 609.

Table 10-9. PWL Function Types

PWL(x1, y1, ... xn, yn**)**	Generates a piece-wise linear function. xn and yn are used to calculate the equivalent output value. This standard PWL function plots all specified points.

Table 10-9. PWL Function Types

PWL_CTE(x1, y1, ... xn, yn**)**	Generates a piece-wise linear function. `xn` and `yn` are used to calculate the equivalent output value. This function plots all specified points but will hold the first specified value if the simulation start time is less than the first time point specified in the function. Similarly it will hold the last value until the simulation is complete.
PWL_LIN(x1, y1, ... xn, yn**)**	Generates a piece-wise linear function. `xn` and `yn` are used to calculate the equivalent output value. This function linearly extrapolates the first value using the first two points of the function. The last value will also be linearly extrapolated using the last two points. This will only happen if the simulation start time is less than the first time value specified in the function, and the simulation end time is greater than the last time value of the function.

Notes

- Waveform definitions can be specified inside subcircuit definitions. Identical complex calculations can be performed for each instance of the subcircuit. Typically the subcircuit is declared once and instanced many times. This means there is no need to have many instances of the same subcircuit with definitions of **`.DEFWAVE`** and **`.PLOT`** for each. Eldo only calculates the **`.DEFWAVE`** statements that are called, not all **`.DEFWAVE`** statements declared.
- Waveform names cannot contain arithmetic operators.
- Waveform names cannot contain the hierarchical separator: the period character “`.`”.
- Only one waveform may be defined per **`.DEFWAVE`** statement.
- It is not possible to use functions which expect at least one waveform as input.
- Built-in PPL functions can not be used in Tcl procs that are called from **`.DEFWAVE`** statements.
- A wave is instantiated by preceding the wave name `w`, which can be followed by a format suffix for complex waveforms, for example: `wm`, `wdb`, `wp`, `wi`, `wr` are enclosed in parentheses `()`.
- A waveform expression cannot contain a division by zero. If this is the case, 1.0×10^{-15} is automatically assigned to a value.
- The **`.DEFWAVE`** command is order dependent in that all components of the waveform expression must have already been defined earlier in the netlist. The following is illegal:

```
.defwave power_vdd=i(v1)*v(v1)
...
v1 in out ...
```

  However, the following is allowed:

```
v1 in out ...
...
.defwave power_vdd=i(v1)*v(v1)
```

- The waveform expression must consist of waves and functions which are related to the type of analysis being performed as described in the **.PLOT**/**PRINT** commands later in this chapter. The following is illegal and will produce false results:

```
.defwave amplification=v(2)/v(1)
...
.plot ac w(amplification)
```

  However, the following is allowed:

```
.defwave amplification=vm(2)/vm(1)
...
.plot ac wdb(amplification)
```

- The **PWL** function in **.DEFWAVE** can not be used in complex expressions. An error will be generated if the **.DEFWAVE** uses the function with other operators or mathematical functions. The following is not accepted:

```
.defwave tran f1 = 1 + pwl(5n,0,7n,5,8n,5,9n,1)
```

  However, the following is allowed:

```
.defwave tran f1 = pwl(5n,0,7n,5,8n,5,9n,1)
```

Examples

```
.defwave power_vdd=v(a)*i(vdd)
...
.extract max(w(power_vdd))
```

The example above defines a waveform `power_vdd` as the product of the waveforms **v(**a**)** and **i(**vdd**)**. The maximum value of the waveform is then extracted and listed to the ASCII output file using the **.extract** command.

```
.defwave w1=v(a)/v(b)
...
.extract integ(w(w1))
.tran 1n 100n
.plot tran w(w1)
```

The example above defines a waveform `w1` as waveform **v(**a**)** divided by waveform **v(**b**)**. The integral value of the waveform is extracted and listed to the ASCII output file using the **.extract** command.

For more information, see ".EXTRACT" on page 637.

```
.defwave powf = 0.5*(vdip(vxx) * CONJ(I(vxx)))
.plot ac wr(powf)
```

Plots the AC power of the `vxx` voltage source.

Example of applying a wave operator:

```
.EXTRACT LABEL = A MAX(V(1))
.EXTRACT LABEL = B MAX(V(2))
.STEP PARAM P1 1 3 1
.DEFWAVE SWEEP my_wave = MEAS(a) + MEAS(b)
.end
```

Example using the **SIGMA** function to sum various items (numbers, parameters or output quantities); when items are output quantities, wildcards are accepted:

```
.EXTRACT dc SIGMA(v(*),1.0,'p1+2')
.DEFWAVE tran star3=SIGMA(ISUB(X2*.g),V(E*))
```

Example of using the **PWL** function:

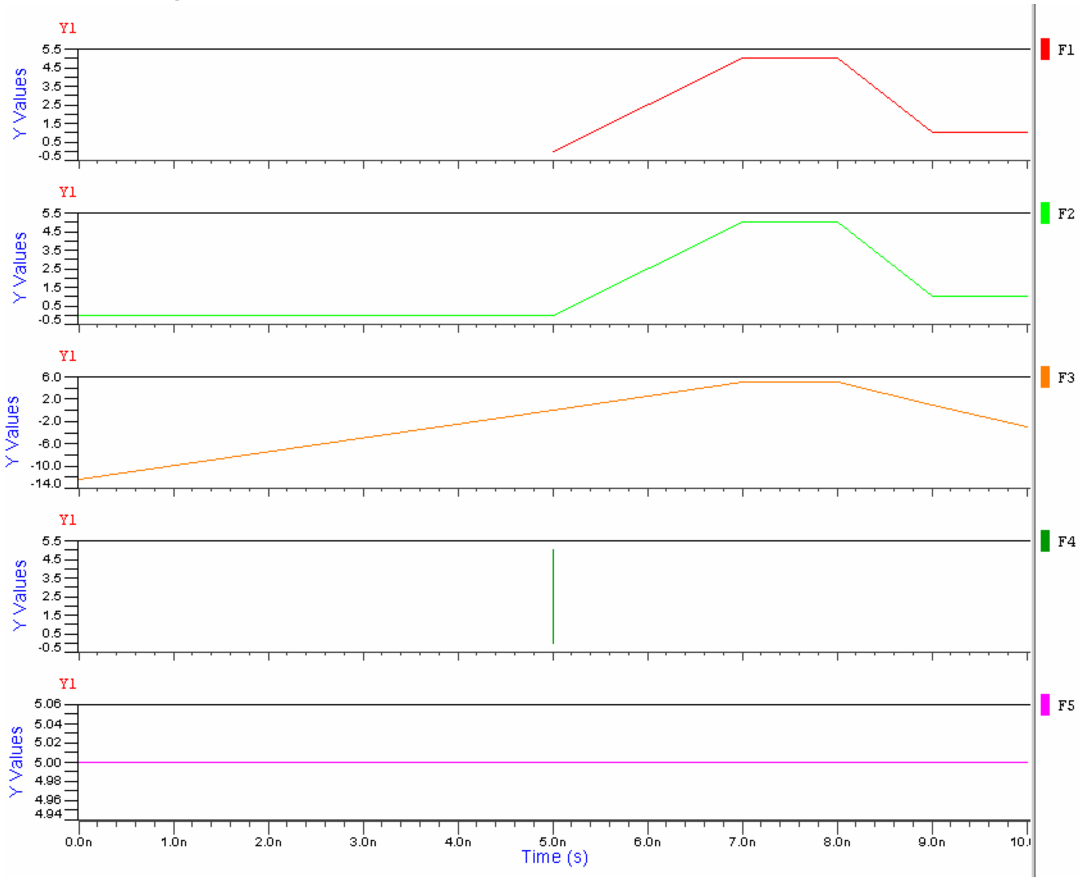
```
v1 1 0 pwl ( 0 0 10n 10)
r1 1 0 1
.defwave tran foo = pwl(0,0,20n, meas(ex))
.extract label = ex yval(v(1),5n)
.tran 1n 10n
.plot tran w(foo)
.end
```

Example of using the **PWL**, **PWL_CTE** and **PWL_LIN** functions.

```
v1 1 0 pwl ( 0 0 10n 10)
r1 1 0 1
.defwave tran f1 = pwl(5n,0,7n,5,8n,5,9n,1)
.defwave tran f2 = pwl_cte(5n,0,7n,5,8n,5,9n,1)
.defwave tran f3 = pwl_lin(5n,0,7n,5,8n,5,9n,1)
.defwave tran f4 = pwl_lin(5n,0, 5n, 5)
.defwave tran f5 = pwl_lin(0n, 5)
.extract label = ex yval(v(1),5n)
.tran 1n 10n
.plot tran w(f1) w(f2) w(f3) w(f4) w(f5)
.end
```

The plot obtained from the netlist is shown in Figure 10-3. The wave `f4` demonstrates how to plot vertical waves with the **.DEFWAVE** command.

Figure 10-3. PWL, PWL_CTE and PWL_LIN Functions



The example below shows how waveform definitions can be specified inside subcircuit definitions. In this example, the *sat* curves for both devices are plotted, even though they are calculated differently.

```
.subckt nmos d g s b
m1 d g s b nmos w=10u l=1u
.defwave sat=vds(m1)-vdss(m1)
.defwave varI=(0.59*gm(m1)*gm(m1)+1.133*ids(m1)*ids(m1))*5u*1u
.ends

.subckt pmos d g s b
m1 d g s b pmos w=5u l=1u
.defwave sat=vdss(m1)-vds(m1)
.defwave varI=(0.79*gm(m1)*gm(m1)+1.589*ids(m1)*ids(m1))*5u*1u
.ends

x0 g g s b nmos
x1 d g s b nmos
x2 d g s b pmos

.plot dc w(x0.sat) w(x1.sat) w(x2.sat)
.defwave Ioffset=sqrt(w(x0.varI)+w(x1.varI)+w(x2.varI))
.plot dc w(Ioffset)
```

.DEL

Remove library name

```
.DEL LIB LIB_NAME
```

This command removes a library name from the nominal description. It can be used in conjunction with the **.ALTER** command to create modifications within the netlist before re-runs.

Please refer to ".ALTER" on page 549.

Example

```
v1 1 0 1
x1 1 2 r
r2  2 0 1
.dc
.extract dc v(2)
.lib delib1.lib
.alter
.del lib delib1.lib
.lib delib2.lib
.end
```

This example demonstrates how a library file can be replaced after a first run. *delib1.lib* is removed and *delib2.lib* is put in its place.

.DEX

Design of Experiments

```
.DEX
+ EXPERIMENT = SCREENING | SCREENING_CTRL | SCREENING_NOISE
+ RESPONSE = LIST_OF_MEASURES
+ [DESIGN = ORTHA_2_N | ORTHA_2_2N | FULL_FACT]
+ [FACTOR = LIST_OF_FACTORS]
+ [FIND_FACTOR]
```

The Eldo command **.DEX** can be used to perform factor screening before attempting to solve subsequent statistical problems. The primary purpose of a variable screening experiment is to select or screen out the few important main effects from the many less important ones.

For the complete description, see the separate chapter Statistical Experimental Design and Analysis.

.DISCARD

Ignore Instances or Subckt Definitions

```
.DISCARD INST | SUBCKT | DEV =(name {,name})
```

This command is used to specify whether one or more subcircuits/instances/devices should be ignored during an Eldo simulation. The **.DISCARD** command cannot remove *hierarchical* instances and cannot contain wildcards.

Parameters

- `INST`

  Specifies that one or more subcircuit instances will be ignored.

- `SUBCKT`

  Specifies that all instances of the subcircuit(s) will be ignored.

- `DEV`

  Specifies that all instances of the primitive element(s) will be ignored.

- `name`

  The name of the instance or subcircuit that will be ignored. A list of names can be specified, in this case the brackets must be used. See Example.

Example

In the example below the instances `X1` and `X2` and the subcircuit `RESI` will be ignored.

```
.SUBCKT RESI a b
R1 a b 1
.ENDS RESI

X1 1 0 RESI
X2 1 0 RESI
X3 1 0 RESI

.DISCARD INST=(X1, X2)
.DISCARD SUBCKT=RESI
```

In the example below the primitive element `r2` will be ignored.

```
*SEARCH extracted value should be -2
i1 1 0 1
r1 1 2 1
r2 1 2 1
r3 2 0 1
.discard dev = r2
.dc
.op
.extract dc v(1)
.end
```

.DISFLAT

Disable Flat Netlist Mode

```
.DISFLAT
```

This command is used to disable the flat netlist mode.

In the case of a large netlist, which does not use any subcircuit instances, Eldo will assume that the netlist is flat and will automatically enter into a mode where it consumes less memory for parsing the netlist. However, this mode requires that all **.MODEL** statements are placed at the top of the netlist.

A message is displayed when Eldo enters this mode. Use the command **.DISFLAT** to disable this mode.

.DISTRIB

User Defined Distributions (Monte Carlo)

```
.DISTRIB DIST_NAME (DEV1 PROB1) [{(DEVn PROBn)}]
```

This command is used to specify user defined distributions for the device tolerances **DEV** and **LOT** used in Monte Carlo analysis. As such, **.DISTRIB** commands are only active when the **.MC** command is present in the netlist causing a Monte Carlo analysis to be performed.

Different entities are able to share the same distribution. Anywhere Eldo accepts **LOT**/**DEV** specifications, you can specify **LOTGROUP**=group_name.

Please refer to ".LOTGROUP" on page 701.

Parameters

- DIST_NAME

  Name of the user defined distribution being specified.

- DEVxx

  A deviance value in the range [–1, 1]. DEVxx values must be ordered from lower to upper values. Two successive values of DEVxx may be equal in the case of steep distributions.

- PROBxx

  A probability value in the range [0, 1].

.DSP

DSP (Digital Signal Processing) Computation

```
.DSP LABEL=label_name MODEL=model_name [DSP=dsp_name] waveform_name
```

Selects the waveform on which the user requires DSP to be applied.

DSP Computation (Extended Syntax)

```
.DSP LABEL=label_name [DSP=dsp_name] waveform_name
+ [TSTART=val] [TSTOP=val] [FS=val] [NBPT=val]
+ [PADDING=val] [WINDOW=name] [ALPHA=val] [BETA=val]
+ [NORMALIZED=val] [INTERPOLATE=val] [DISPLAY_INPUT=val]
+ [FNORMAL=val] [FMIN=val] [FMAX=val]
+ [NAUTO=val] [NCORR=val] [NPSD=val] [NSECT=val]
+ [NBINTERVAL=val
+ [XSTART=val XSTOP=val SAMPLE=YES|NO FS=val]
+ [BASELINE=val] [TOPLINE=val]
+ [EDGE_TRIGGER = RISING | FALLING | EITHER]
+ [XSTART=val] [XEND=val]
```

DSP models can also be specified directly on the **.DSP** command, using the extended syntax, as an alternative to using **.DSP** in conjunction with **.DSPMOD**. See “.DSPMOD” on page 625 for details of the additional parameters.

Parameters

- `label_name`

  Label name to select which wave to be plotted (**.PLOT**), or on which wave quantities have to be extracted (**.EXTRACT**). Required.

- `model_name`

  The name of the computational model, defined in the **.DSPMOD** command, to apply on the wave specified by `waveform_name`. `model_name` must be a valid **.DSPMOD** label. Required.

- `dsp_name`

  Specifies the correlogram, periodogram, histogram, or frequency method.

- `waveform_name`

  Any regular Eldo waveform name: there can be any number of **.DSP** commands specified in the *.cir* file.

Examples

```
.DSP LABEL=LBL1 MODEL=PSD_CORRELO V(1)
.DSP LABEL=LBL2 MODEL=PSD_PERIODO W('v(2)+v(4)')
```

These DSP commands define two new PSD entities named `LBL1` and `LBL2`. `LBL1` is obtained by applying a DSP model `PSD_CORRELO` on wave `V(1)`. `LBL2` is obtained by applying a DSP model PSD_PERIODO on wave V(2)+V(4). PSD_CORRELO and PSD_PERIODO must be defined by a .DSPMOD command.

Display and **.EXTRACT** of DSP results:

```
[.PLOT|.PRINT|.PROBE ] DSP DSPxx(label_name)
.EXTRACT DSP <expression>
```

where xx stands for DB, R, I, P, M.

```
.DSP LABEL=LBL2 MODEL=PSD_PERIODO V(1)
.PLOT DSP DSPDB(LBL2)
```

It is also possible to extract values from a DSP waveform:

```
.DSP LABEL=LBL1 MODEL=PSD_CORRELO V(1)
.PLOT DSP DSPDB(LBL1)
.EXTRACT DSP YVAL(DSPDB(LBL1),5MEG)
```

Eldo will print out the value (in dB) at 5 Meg for the DSP done on V(1).

The following is an example of the extended syntax, the following line:

```
.DSP LABEL=FOO DSP=FREQUENCY XSTART=20n XSTOP=100n v(out)
```

is equivalent to the Frequency Computation **.DSPMOD** example:

```
.DSPMOD LABEL=FMOD DSP=FREQUENCY XSTART=20n XSTOP=100n
.DSP LABEL=FOO MODEL=FMOD v(out)
```

.DSPF_INCLUDE

Load DSPF File

```
.DSPF_INCLUDE [FILE=]DSPF_FILENAME [INST={list_of_subckt_inst}]
+ [LEVEL=C|RC|RCC] [DEV=SCH[EMATIC]|DSPF] [RMINVAL=val] [CMINVAL=val]
+ [CCMINVAL=val] [ADDXNET] [ADDX] [MSUFFIX=string]
.DSPF_INCLUDE [FILE=]DSPF_FILENAME [LEVEL=C|RC|RCC]
+ [DEV=SCH[EMATIC]|DSPF] DEDICATEDX=subckt_name [RMINVAL=val]
+ [CMINVAL=val] [CCMINVAL=val] [ADDXNET] [ADDX] [MSUFFIX=string]
```

By specifying this command inside a netlist, Eldo will add the parasitic elements described inside the specified DSPF (Detailed Standard Parasitic Format) file.

Use the option **MAX_DSPF_PLOT**=val (“MAX_DSPF_PLOT=VAL” on page 1003) to override the maximum number of DSPF interface nodes plotted. By default, this limit is 100.

Use option **KEEP_DSPF_NODE** (“KEEP_DSPF_NODE” on page 1002) to force Eldo to keep the original **.PLOT**/**.PROBE** command if a node coming from the DSPF has the same name. When a node is replaced by a parasitic net using a DSPF file, this node can be renamed or even removed from the design.

The **INST** and **DEDICATEDX** specifications are mutually exclusive.

See also commands .IGNORE_DSPF_ON_NODE and .SELECT_DSPF_ON_NODE for additional control over which nodes should be ignored or selected for parasitics.

Specify option **COLLAPSE_DSPF_OUTPUT** (“COLLAPSE_DSPF_OUTPUT[=@|#]” on page 999) to force Eldo to regroup currents (I, Ix, Isub) and noise outputs according to the device or subckt instance specified in arguments. By default the DSPF character is set to @.

Parameters

- DSPF_FILENAME

  Name of the DSPF file. This parameter must be the first specified on the command line. **FILE=** is optional.

- **INST**

  Specifying this parameter will force Eldo to only include the parasitic elements for the specified subcircuit instances given in list_of_subckt_instances. It will assume that the DSPF file is hierarchical.

- **LEVEL**=**C**|**RC**|**RCC**

  Defines the level of parasitics to use from the specified DSPF file. This can also be specified using option **DSPF_LEVEL** (see “DSPF_LEVEL=C|RC|RCC” on page 976 for more information). Local filter of the DSPF file, level can be:

  C—backannotation of total (lumped) net capacitance contained in the DSPF file

RC—backannotation of distributed R/C/L contained in the DSPF file, ignoring floating capacitors

RCC—backannotation of any distributed R/C/CC/L/K contained in the DSPF file (default)

- **DEV=SCH[EMATIC]|DSPF**

  SCH[EMATIC] specifies the DSPF file contains only the parasitics. Default.

  DSPF specifies the DSPF file contains both the parasitics and the intentional devices. Sometimes it is more accurate to have the extractor generating a file containing both. For this sort of file, **.SELECT_DSPF_ON_NODE** and **.IGNORE_DSPF_ON_NODE** are ignored.

  When **DEV=DSPF** is specified, Eldo will take three actions:

  - Eldo will include the file specified in the command as if it was a **.LIB** command. Eldo will therefore have two variants of the same subcircuit in its database, one corresponding to the intentional devices which should be present in the circuit, and another one included by the **.DSPF_INCLUDE** command, corresponding to the version with its parasitics.
  - Eldo will emulate a **.BIND** command on the instances/subcircuits which are specified on the **.DSPF_INCLUDE** command.
  - Eldo will apply filters, if any, to the DSPF devices.

  Generally, the number of pins and the pin order given in the **.SUBCKT** found in the DSPF file differs from the list in the intentional design. Eldo will check for pin names in both intentional and DSPF subcircuit to ensure proper connection, based on node name.

  If the DSPF version contains more pins than the intentional, then it is assumed that these pins correspond to global nets, and Eldo will make map as such. Errors will be issued if no such global name can be found. Note that the index of the pin in the IX or VX plot command refers to the index of the intentional design.

- **RMINVAL**

  Specifies the minimum resistance of a resistor, in Ω. Resistors with a value less than the specified minimum value are removed from the netlist and replaced by short circuits. The default value is $-\infty$.

- **CMINVAL**

  Specifies the minimum value of a grounded capacitor, in Farads. Grounded capacitors with a capacitance of less than the specified minimum value are removed from the netlist. The default value is $-\infty$.

- **CCMINVAL**

  Specifies the minimum value of a floating capacitor, in Farads. Floating capacitors with a capacitance of less than the specified minimum value are removed from the netlist. The default value is $-\infty$.

Note

The reason for the –∞ defaults (instead of zero) is to take into account the possibility of negative resistances.

- **ADDXNET**

  Keyword instructing Eldo to insert an X character on hierarchical names specified in the DSPF file for retrieving the node in the pre-layout design (where X is added as necessary). This addition of the X character is used only for the *| NET instructions. For an example, see “Example 3” on page 624.

- **ADDX**

  Keyword instructing Eldo to insert an X character on hierarchical names specified in the DSPF file for retrieving the node in the pre-layout design (where X is added as necessary). This addition of the X character is used only for both *| NET and *| I instructions. There is no flag specifically for just *| I instructions.

- **MSUFFIX**=string

  Can be specified to handle situations where a device in the nominal design is split into different devices in the DSPF. Usually MSUFFIX is the “__” or the “@” string. The use of the string depends on the value specified for DEV.

  - **DEV=SCHEMATIC**

    In this mode, the DSPF file contains only the parasitics (R, L, C devices), which will be added to the nominal design.

    ```
    *|NET N$26 7.87008e-14
    *|I (X_X2:minus X_X2 minus B 0.0 49.165 301.725)
    *|I (X_X14:g X_X14 g I 0.0 55.38 255.93)
    *|I (X_X14__2:g X_X14__2 g I 0.0 55.38 277.71)
    ```

    It can happen that the nominal design contains only one instance (such as X_X14 in the example above), while in the DSPF this device is split in several devices, and the name of these devices is built by catenating to the nominal name a predefined string, which is usually “__” or “@”, followed by an index.

    For instance, X_X14__2 in the example above derives from X_X14.

    By default, Eldo will complain that it is not able to retrieve the devices with the extensions, and for sake of security does not apply any DSPF insertion on the node. But if MSUFFIX is set, Eldo will handle this situation by simply connecting the nodes which appear in the extended devices to their counterpart in the nominal device.

    With the example above, Eldo will connect X_X14__2:g to the node X_X14:g and will process the DSPF accordingly.

    This means that Eldo will re-catenate the devices, hence simplifying the DSPF topology.

- **DEV=DSPF**

  In this mode, the DSPF file contains both the nominal devices and the parasitics. The only problem here is a problem of printout, as illustrated below:

```
X_X24 X_X24:PLUS X_X24:MINUS VMINUS RPO1PM1 R=533.401 w=4e-06 NC=11 lpe=3
X_X24@3 X_X24@3:PLUS X_X24@3:MINUS VMINUS RPO1PM1 R=533.401 w=4e-06 NC=11
+ lpe=3
X_X24@2 X_X24@2:PLUS VMINUS VMINUS RPO1PM1 R=533.401 w=4e-06 NC=11 lpe=0
```

  if there is a printout such as .PLOT IX(X_X24.1), then results will not be correct since the output will not take into account the current flowing devices X_X24@3 and X_X24@2.

  Therefore, if MSUFFIX is set to "@", then Eldo will recognize that X_X24@3 and X_X24@2 do derive from X_X24 (X_X24 must appear first in the DSPF files, and the 'child' must follow), and Eldo will then replace the 3 X lines above by the statements:

```
X_X24 X_X24:PLUS X_X24:MINUS VMINUS RPO1PM1 R=533.401 w=4e-06 NC=11 lpe=3
M = 3
.connect X_X24:PLUS X_X24@3:PLUS
.connect X_X24:PLUS X_X24@2:PLUS
.connect X_X24:MINUS X_X24@3:MINUS
.connect X_X24:MINUS X_X24@2:MINUS
```

  This means that when MSUFFIX is set, corresponding devices are merged again into a single device with the proper M factor value. This means that DSPF is simplified, but output commands are providing correct results. Note that with the scheme above, in case of mismatch of parameters between nominal and subsequent instances, Eldo will take the parameter values of the nominal statement: warning will be then displayed.

- **DEDICATEDX**=subckt_name

  The specified DSPF file, will *only* be used for the dedicated instance `subckt_name`. Eldo will assume that the **.SUBCKT** command is not specified in the DSPF file and that the node name in the DSPF file refers to the node names local to the dedicated instance name. Without the option Eldo will assume that the nodes/objects in the DSPF are the top level nodes/objects.

In DSPF files, it is common not to specify the `X` on subcircuit names that are referenced in the file. When Eldo cannot find a NET name (in `*|NET` command), Eldo will check whether the NET exists with the letter `X` specified. If the node is found, Eldo will assume that the `X` was meant for all node names in the DSPF file.

If the contents of the DSPF file are:

```
.subckt top IN OUT
*|NET TOP/net1 0.827ff
*|I (TOP/1/M1:D TOP/1/M1 D O 0ff)
*|I (TOP/1/M2:D TOP/1/M2 D O 0ff)
*|I (TOP/2/M1:G TOP/2/M1 G I 0.070ff)
*|I (TOP/2/M2:G TOP/2/M2 G I 0.105ff)
R1 TOP/1/M1:D TOP/net1:1 100
```

and the contents of the *.cir* file are:

```
.subckt top in out
XTOP in out inv2
.ends
V1 in 0 pulse( 0 2 1p 100p 100p 5n 10n)
C1 out 0 1f
X1 in out top
```

Eldo will not find the node `X1.TOP.NET1`. Therefore, Eldo will look for `X1.XTOP.NET1`, which will be found. Eldo will now *assume* that the letter `X` should be used for all the nodes in the DSPF. Eldo will now assume that the instruction:

```
R1 TOP/1/M1:D TOP/net1:1 100
```

should be:

```
X1.XTOP.NET1_R1 X1.XTOP.X1.M1:D X1.XTOP.NET1:1 100
```

Connecting a source to a DSPF back-annotated node

It is possible to connect a source to a node that is back-annotated with DSPF. This might be useful to insert a voltage or current source to a specific node to check a critical path. However, after DSPF extension, the original node might be expanded in several nodes, and there might be no path remaining between the extra source and the network, that is, the extra source no longer has a connection to the network. This is because the extra source was not in the design when DSPF information was created. The solution is that nodes connected to that extra source are not expanded by DSPF. Eldo can detect such situations when the extra source is instantiated from the top of the design and connected to a node at a lower level of hierarchy, using the full node pathname. For example:

```
.SUBCKT INV in out
...
R1 in internal 1k
...
.ends
X1 in out INV
v1 X1.internal 0 5    !full pathname usage
```

Eldo will detect this configuration, and DSPF instructions will be ignored on node X1.internal. This is similar to the command: **`.IGNORE_DSPF_ON_NODE`** `X1.internal`.

However if the extra source is added at the same level as the nodes (for example all at the top), Eldo will only be able to issue a message that the node is connected to a single device only, and will indicate that there is a possible DSPF connection problem. The user must then manually add the command **.IGNORE_DSPF_ON_NODE** on that node in order to deal with that specific situation.

Related Commands

.IGNORE_DSPF_ON_NODE, see ".IGNORE_DSPF_ON_NODE" on page 684,
.MAP_DSPF_NODE_NAME, see ".MAP_DSPF_NODE_NAME" on page 705,
.SELECT_DSPF_ON_NODE, see ".SELECT_DSPF_ON_NODE" on page 862.

Example

This example (see *test_dspf.cir* netlist below) can be run with and without the following line to compare the simulation results with or without the parasitics effects:

```
.dspf_include testspf2.spf
```

Inside this example, when the DSPF is included, the node `XTOP.net1` is replaced by a network of resistors, so, according to the DSPF file, the node itself does not exist anymore, but is replaced by three different nodes: `XTOP.net1:1`, `XTOP.net1:2`, `XTOP.net1:3`. In this case, the plot statement: **.plot tran v**`(in)` **v**`(out)` must be replaced by **.probe tran v**`(XTOP.net1*)` to plot all the nodes "composing" the original node `XTOP.net1`.

File *test_dspf.cir* (located in *$MGC_AMS_HOME/examples/eldo/*):

```
* test DSPF
.dspf_include testspf2.spf

.global vdd gnd

.subckt inv a y
+ln=2e-07 wn=2.8e-07 lp=2e-07 wp=2.8e-07
M1 Y A vdd vdd EPLLMM9JU w=wp l=lp m=1
M2 Y A gnd gnd ENLLMM9JU w=wn l=ln m=1
.ends

.subckt inv2 a y
X1 a net1 inv ln=0.18u wn=2.0u lp=0.18u wp=4.0u
X2 net1 y inv ln=0.18u wn=2.0u lp=0.18u wp=4.0u
.ends

V1 in 0 pulse( 0 2 1p 100p 100p 5n 10n)
C1 out 0 1f
XTOP in out inv2
VDD  vdd 0 2
VGND gnd 0 0

.plot tran v(in) v(out)
*.plot tran v(XTOP.net1)
.probe tran v(XTOP.net1*)

.tran 1n 20n
```

```
.model EPLLMM9JU PMOS level=53 rd=100 rs=100
.model ENLLMM9JU NMOS level=53 rd=80 rs=80

.end
```

File *testspf2.spf*:

```
*|DSPF 1.0
*|DIVIDER /
*|DELIMITER :
*|GROUND_NET VSS
*|NET XTOP/net1 0.827ff
*|I (XTOP/X1/M1:D XTOP/X1/M1 D O 0ff)
*|I (XTOP/X1/M2:D XTOP/X1/M2 D O 0ff)
*|I (XTOP/X2/M1:G XTOP/X2/M1 G I 0.070ff)
*|I (XTOP/X2/M2:G XTOP/X2/M2 G I 0.105ff)
R1 XTOP/X1/M1:D XTOP/net1:1 100
R2 XTOP/X1/M2:D XTOP/net1:1 100
R3 XTOP/net1:1 XTOP/net1:2 2000
R4 XTOP/net1:2 XTOP/net1:3 2000
R5 XTOP/X2/M1:G XTOP/net1:3 100
R6 XTOP/X2/M2:G XTOP/net1:3 100
C1 XTOP/X2/M1:G VSS 1.53ff
C2 XTOP/X2/M2:G VSS 1.71ff
C3 XTOP/net1:1 VSS 2.80ff
C4 XTOP/net1:2 VSS 2.80ff
C5 XTOP/net1:3 VSS 2.80ff
.END
```

Parasitic resistor values have been multiplied by 1000 in order to display a real DSPF loading impact on the results.

Example 2

When the DSPF file does not contain the **.SUBCKT** command, for example:

```
dedicated_example.spf
*|NET TOP/net1 0.827ff
...
```

and the *.cir* file contains:

```
.SUBCKT top in out
XTOP in out inv2
.ENDS
V1 in 0 pulse( 0 2 1p 100p 100p 5n 10n)
C1 out 0 1f
X1 in out top

.DSPF_INCLUDE dedicated_example.spf DEDICATEDX=X1
```

Specifying **DEDICATEDX=X1**, the node **TOP** in the DSPF file will refer to the node **X1.TOP** in the *.cir* file, that is, node **IN**.

Example 3

Naming in DSPF files can vary from one extractor to the next. In particular, node names specified in the *|NET instructions and device names specified in the *|I instructions may or may not contain the letter X in names.

For instance, inside the DSPF could be:

```
*|NET I117/net18 0.00415015PF
*|I (XI117/XM26:D XI117/XM26 D   B 0 62.255 143.84)
*|I (XI117/XM31:D XI117/XM31 D   B 0 67.595 146.295)
```

This means that on *| NET is XI117/net18, while on the *|I instance naming is correct from the Eldo point of view.

Setting keyword parameter **ADDXNET** instructs Eldo to insert an X character on hierarchical names specified in the DSPF file for retrieving the node in the pre-layout design (where X is added as necessary).

.DSPMOD

PSD (Power Spectral Density) Computation

```
.DSPMOD DSP=CORRELO|PERIODO LABEL=label_name
+ [TSTART=val] [TSTOP=val] [FS=val] [NBPT=val]
+ [PADDING=val] [WINDOW=name] [ALPHA=val] [BETA=val]
+ [NORMALIZED=val] [NORMALIZATION_FACTOR=val] [INTERPOLATE=val]
+ [DISPLAY_INPUT=val] [FNORMAL=val] [FMIN=val] [FMAX=val]
+ [NAUTO=val] [NCORR=val] [NPSD=val] [NSECT=val]
```

For PSD parameter descriptions, please see "PSD Computation Parameters" on page 626. For details on the principles of PSD computation, see Power Spectral Density in the *EZwave User's and Reference Manual*.

Two methods are available for calculating this quantity. The Correlogram Method means that the FFT is used directly to compute estimates of the PSD function $Rxx(n)$ for N_{auto} lags, where $2 \times N_{auto}$ is the size of the transform used. The Periodogram Method means that a sliding FFT is used to compute estimates of the PSD directory rather than estimating a PSD function.

Histogram Computation

```
.DSPMOD DSP=HISTOGRAM LABEL=label_name NBINTERVAL=val
+ [XSTART=val XSTOP=val SAMPLE=YES|NO FS=val]
```

Generates a histogram of the input wave showing the wave's magnitude probability density distribution. The input wave is first sampled to equidistant points if the optional argument `SAMPLE` was set.

For Histogram parameter descriptions, please see "Histogram Computation Parameters" on page 628.

Frequency Computation

```
.DSPMOD DSP=FREQUENCY LABEL=label_name
+ [BASELINE=val] [TOPLINE=val]
+ [EDGE_TRIGGER = RISING | FALLING | EITHER]
+ [XSTART=val] [XEND=val]
```

Provides access to the frequency measurement of EZwave. It returns the instantaneous frequency of a signal. The measurement is not performed by Eldo, the default values for the parameters are defined by EZwave. Using this DSP function forces the use of the JWDB server, even if JWDB output has been disabled.

For Frequency parameter descriptions, please see "Frequency Computation Parameters" on page 629.

There can be any number of **.DSPMOD** commands specified in the *.cir* file.

A more compact syntax is also available. It allows to define the model parameters directly on the **.DSP** command, without the need for a separate **.DSPMOD** statement. For example, the following line is equivalent to the Frequency Computation Example:

```
.DSP LABEL=FOO DSP=FREQUENCY XSTART=20n XSTOP=100n v(out)
```

See ".DSP" on page 615 for more details.

PSD Computation Parameters

- **DSP**=**CORRELO**|**PERIODO**

  Specifies either the correlogram or periodogram method.

- **LABEL**

  Defines the name of the computational model, which will then be selected in the **.DSP** command to be applied to a specific input signal. Required.

- **TSTART**

  Time value from which the time window will start.

- **TSTOP**

  Time value that specifies the end of the time window.

- **FS**

  Sampling frequency.

- **NBPT**

  Number of sampling points to be used in the time window.

  These parameters satisfy the following equation:

  $$\frac{Nbpt - 1}{Fs} = Tstop - Tstart$$

- **PADDING**

  Specifies that padding zeros before or after the signal can be added.

  0 = No padding

  1 = Zeros are added at the beginning of the FFT input window

  2 = Zeros are added at the end of the FFT input window (default)

  3 = Zeros are added evenly at the beginning and at the end of the FFT input window.

- **WINDOW**

  Specifies the Sampling Window to be used. The following parameters are allowed: **RECTANGULAR** (default), **PARZEN**, **BARTLETT**, **WELCH**, **BLACKMAN**, **BLACKMAN7**, **KLEIN**, **HAMMING**, **HANNING**, **KAISER**, **DOLPH_CHEBYCHEV**.

For further details, please see the *EZwave User's Manual*.

- **ALPHA**

  Used when the **WINDOW** is **DOLPH_CHEBYCHEV** or **HANNING**.

- **BETA**

  Used when the **WINDOW** is **KAISER**. Constant which specifies a frequency trade-off between the peak height of the side lobe ripples and the width of energy in the main lobe.

- **NORMALIZED**

  Specifying `1` means that all the points of the result are multiplied by 2/**NBPT**. Default. Specifying `0` means no normalization of the results.

- **NORMALIZATION_FACTOR**

  Replaces the default factor (2/**NBPT**) which is applied to FFT results when **NORMALIZED** is set to 1. It is ignored if **NORMALIZED** is set to 0.

- **INTERPOLATE**

  Sets type of interpolation:

  0 = No interpolation (default)

  1 = Linear interpolation

  2 = Cubic Spline interpolation

  3 = Blocker Sampler interpolation

- **DISPLAY_INPUT**

  Allows visualization of the transient window used as input of the PSD: this waveform contains equally-spaced time value points.

  0 = do not display PSD input (default)

  1 = display PSD input (if output is displayed)

- **FNORMAL**

  Adjusts the results around the Y-axis so that the point for the specified frequency is 0.0.

- **FMIN**

  Starting frequency used inside the result window.

- **FMAX**

  Last frequency used inside the result window.

- **NAUTO**

  Number of points for the autocorrelation result.

- **NCORR**

  Number of autocorrelation points used for the PSD computation.

- **NPSD**

  Number of points for the PSD result.

- **NSECT**

  Number of points for each section. Can only be specified if the periodogram method is specified (**DSP**=PERIODO).

PSD Computation Examples

```
.DSPMOD DSP=CORRELO LABEL=PSD_CORRELO
+ TSTART = 200n
+ TSTOP = 400n
+ NORMALIZED = 0
+ INTERPOLATE=0
+ DISPLAY_INPUT=1
```

This DSP model command defines a set of parameters for a PSD computation using the correlogram method. Input values will be sampled from 200ns to 400ns. Default value for number of sampling points (NBPT) is 1024. Output signals will not be normalized and the PSD input signal will be dumped in the output file.

```
.DSPMOD DSP=PERIODO LABEL=PSD_PERIODO
+ TSTART = 0
+ TSTOP = 1.93e-8
+ NBPT = 100
+ NORMALIZED = 0
+ INTERPOLATE=0
+ DISPLAY_INPUT=1
+ FS=5.12e9
+ ALPHA=0.5
```

This DSP model command defines a set of parameters for a PSD computation using the periodogram method. The input signal has 100 points, sampled between 0 and 19.3 ns.

Histogram Computation Parameters

- **DSP=HISTOGRAM**

  Specifies the histogram method.

- **LABEL**

  Defines the name of the computational model, which will then be selected in the **.DSP** command to be applied to a specific input signal. Required.

- **NBINTERVAL**

  Specifies the number of intervals which will be used to compute the distribution.

- **XSTART**

  X value from which the measurement window will start.

- **XSTOP**

  X value from which the measurement window will stop.

- **SAMPLE**

  If `YES`, perform a sampling of the input wave (default is `NO`).

- **FS**

  Sampling frequency.

Histogram Computation Example

```
.DSPMOD DSP=HISTOGRAM LABEL=HISTO1 NBINTERVAL=10
+ XSTART=3n XSTOP=END SAMPLE=YES
.DSP label=s_histo model=HISTO1 v(1)
.PLOT dsp dsp(s_histo)
```

This DSP model command defines a set of parameters for a histogram computation, labelled HISTO1. The measurement window is between 3ns and the end of the simulation, with sampling performed on the input signal. The resulting histogram contains ten intervals.

Frequency Computation Parameters

- **DSP**=**FREQUENCY**

  Specifies the frequency method.

- **LABEL**

  Defines the name of the computational model, which will then be selected in the **.DSP** command to be applied to a specific input signal. Required.

- **BASELINE**

  Y-value that sets the low threshold of the signal. If **BASELINE** is not provided then this value will be computed.

- **TOPLINE**

  Y-value that sets the high threshold of the signal. If **TOPLINE** is not provided then this value will be computed.

- **EDGETRIGGER**=**RISING|FALLING|EITHER**

  Type of edge used for the calculation of the frequency. If **EITHER** is specified, there will be no difference made between rising and falling edges.

- **XSTART**

  X value at the beginning of the measurement interval. If the X value is not provided then the X start of the input signal will be used.

- **XSTOP**

  X value at the end of the measurement interval. If the X value is not provided then the X end of the input signal will be used.

Frequency Computation Example

```
.DSPMOD LABEL=FMOD DSP=FREQUENCY XSTART=20n XSTOP=100n
.DSP LABEL=FOO MODEL=FMOD v(out)
```

This DSP model command defines a set of parameters for a frequency computation. The measurement window is between 20ns and 100ns.

.END

End Eldo Netlist

.END

This command terminates the simulator input description. Any text which follows it is ignored, unless there is another **.END** statement; in which case, all lines between the preceding **.END** and the current **.END** will be considered as another circuit to simulate. This enables the possibility to have several circuits in the same netlist file.

For multi-circuit simulations, Eldo simulates all the circuits contained inside the input files; the **.END** statement is optional inside the first circuit, but it must be specified inside the subsequent circuits.

Example

```
first title
...
...
.end
second title
...
...
.end
```

The line following the **.end** statement is the title of the next circuit.

.ENDL

End Eldo Library Variant Description

```
.ENDL
```

This command terminates a library variant description. Any text which follows it is ignored.

See "Library variant management" on page 694 for details.

.ENDS

End Eldo Subcircuit Description

```
.ENDS
```

This command terminates an Eldo subcircuit description. Any text which follows it is ignored.

Note

.EOM is equivalent to **.ENDS**.

.EQUIV

Replace Node Name for Display

```
.EQUIV new_name=netlist_name
```

This command is used to change the name of a netlist node to a new display name. The new name can then be used in display output statements such as **.PLOT** and **.EXTRACT**. Both the original name and the new name will be valid. The name can be a hierarchical name.

Parameters

- `new_name`

  New name of node for display.

- `netlist_name`

  Node name to be changed.

Examples

This is a full netlist using the **.EQUIV** command:

```
equiv

.sigbus B[3:0] vhi=2.5 vlo=0 trise=0.1n tfall=0.1n base=hexa
+  0n A
+ 10n 3
+ 20n 4
+ 30n 8
+ 50n F
r0 B[0] 0 1k
r1 B[1] 0 1k
r2 B[2] 0 1k
r3 B[3] 0 1k

.tran 1n 60n
.equiv enable=b[0]
.equiv xin.out=b[1]

.probe
.plot tran v(enable)
.plot tran v(xin.out)
```

From the above netlist, the following two lines change the names of the nodes `b[0]` and `b[1]` to `enable` and `xin.out`. The name can be anything the user chooses.

```
.equiv enable=b[0]
.equiv xin.out=b[1]
```

From the above netlist, the **.PLOT** command uses the new node names, `v(enable)` and `v(xin.out)`. These names will be the names displayed in the waveform viewer after the simulation is carried out.

```
.plot tran v(enable)
.plot tran v(xin.out)
```

The following example uses the **.EQUIV** command with a voltage source node:

```
v1 vss 0 5
.tran 10n 100n
.equiv vin=vss
.plot tran v(vin)
```

The following example show how the new name can be a hierarchical name:

```
.EQUIV XBUF.IN = NET$34
.PLOT TRAN V(XBUF.IN)
```

.EXTMOD

Extract Mode

```
.EXTMOD FILE=filename
```

The **.EXTRACT** commands are merged with the simulation itself. In some cases, this causes a lot of wasted CPU time. If the user makes a mistake when specifying the **.EXTRACT** command, the command must be fixed, then the simulation itself re-run to obtain the correct results. When the simulation takes a few seconds, this is not a problem. When the simulation takes a few hours of CPU, this is of course a big problem.

Use the **.EXTMOD** command to perform only the **.EXTRACT** commands, without performing the simulation(s). Eldo will decorrelate the ‘simulation’ from the ‘extraction’. See “.EXTRACT” on page 637 for more information.

This may cause large output files. Eldo automatically identifies the waveforms used in **.EXTRACT**/**.MEAS** statements and save them to the file. This is preferable to being forced to re-run hours of CPU because the wrong **.EXTRACT** formula was entered.

This functionality can also be specified with the command line argument **-extract filename**, see “-extract filename” on page 47.

- **filename**

  Name of an EZwave *.wdb* output file. Required. The extractions in the netlist will be solved using the waves contained in this file.

.EXTRACT

Extract Waveform Characteristics

```
.EXTRACT [EXTRACT_INFO] [LABEL=NAME] [FILE=FNAME] [UNIT=UNAME] [VECT]
+ [CATVECT] $MACRO|FUNCTION [OPTIMIZER_INFO] [MC_INFO] [TIME=VALUE]
+ [INTERP_MODE=LINEAR|QUADRATIC|SAMPHOLD|HISTOGRAM|SPECTRAL]
```

This command extracts waveform information using a combination of arithmetic expressions or pre-defined functions. A flexible language exists in Eldo to extract characteristics from raw simulation results (for example the maximum value of a waveform, or the time at which a given threshold is crossed by a waveform, and so on). The type of analysis for which the specified extraction is carried out may be defined. This is useful when different types of analyses are performed in the same simulation run.

The results are listed to the ASCII output (*.chi*) file. They can also be written to a specified file *fname.aex* when option **AEX** is specified in the netlist. Use the option **ALIGNEXT** to specify Eldo to write tabulated **.EXTRACT** results in the *.aex* file. By default the results are not tabulated (aligned).

The **.PLOT** command may also be used to plot *extraction* results. It is used to specify which simulation results have to be kept by the simulator for graphical viewing and post-processing. See ".PLOT" on page 791.

When extraction statements are combined with parameter sweeping statements (see ".STEP" on page 890), Eldo automatically creates waveforms showing the extraction results versus the swept parameter. For example, a user may extract the width of an output pulse from a transient simulation, and sweep the power supply level. In this case Eldo will automatically create a waveform showing the width of the pulse versus the power supply level. Similarly, if extractions and **.ALTER** statements are combined, Eldo will automatically create waveforms showing the extraction results versus the index of the **.ALTER** runs (see ".ALTER" on page 549). In this case, the X axis will contain 1, 2, 3, and so on. The initial display in the viewer can also be prepared using **.PLOT** commands (see the **.PLOT EXTRACT**... description, "EXTRACT" on page 798). Example:

```
.EXTRACT TRAN label=VMAX  MAX(V(out))
.PARAM powersupply=1.2
VDD VDD 0 'powersupply'
.STEP PARAM powersupply list 1.2V 1.3V 1.4V 1.6 2V
.PLOT EXTRACT meas(VMAX) ! this will create a waveform
*                          showing VMAX(powersupply)
```

By default, the extraction waveforms, generated by a **.EXTRACT** combined with a sweep (**.TEMP**, **.STEP**, or **.ALTER**), are saved inside the *EXT* folder in the main *.wdb* file, which can be read by EZwave.

To save single datapoint extract results (for example, scalars) in the *.wdb* file, so that they can be compared between *.wdb* files, use option **DUMP_EXTRACT**.

It is also possible to extract values from an FFT waveform. Additionally, in the case where a set of simulation runs were done with a parameter (P) being varied, and a given output (O) extracted, users may wish to extract a value based on the obtained P/O curve.

It is always preferable to insert any **.EXTRACT** commands at the end of the netlist, since they can refer to objects (such as voltage sources) which need to be defined beforehand.

If you make a mistake when specifying a **.EXTRACT** command then you can correct the offending command and re-invoke Eldo using the **-extract** flag (**eldo** <*netlist.cir*> **-extract**). This will only re-run the **.EXTRACT** commands in the netlist without performing the simulation(s), see "-extract filename" on page 47 for more information. This is especially useful for simulations that take a considerable amount of time. This functionality can also be specified using the command **.EXTMOD**, see ".EXTMOD" on page 636 for more details.

There are special extensions to the **.EXTRACT** command for the purpose of optimization.

For more information see the Optimizer in Eldo chapter.

Extract accessor functions, **EXTRACT()**, **MEAS()**, or **INITIAL_EXTRACT()** can be used in **.EXTRACT** statements to perform an "extract of extract", see "Extract accessor functions" on page 643.

To compare an extracted wave to a reference specify the **UNIT** parameter to define the unit of the extract. By default EZwave displays separate axes. This can be used when plotting the extract wave during a sweep analysis.

If a **.EXTRACT** command is present inside a **.ALTER** command, an *EXT* folder will be created in the *.wdb* file on the condition that the **.EXTRACT** statement remains unchanged for each **.ALTER** file. If this condition is not met, the *EXT* folder is not created. For further details, see ".ALTER" on page 549.

You can use wildcards in **.EXTRACT** commands. For example:

```
.extract tran max(v(*))
.extract tran yval(v(x*.[1-9]*),20n)
.extract tran yval(id(M*o),20n)
.extract tran min(i(X*.M*.d),20n)
.extract tran yval(v(x*.ti*),20n)
```

Quantities which allow wildcards are the same as those that can be used in **.PROBE**, see ".PROBE" on page 838. The **-R** flag can be specified with wildcards so that all internal and external nodes of subcircuits beginning with the string specified will be probed, see description of **.PROBE -R** in "Using Wildcards in Subcircuit Instances" on page 843.

The **.EXTRACT** command cannot be used on digital signals.

By default, extraction results are saved inside the *EXT* folder in the main *.wdb* file which can be read by EZwave. Use option **EXTFILE** to create a *.ext.wdb* file with the extraction results. The *.ext.wdb* file will not always be created as it depends on the type of simulation and the specification of the **.EXTRACT** command.

Backward compatibility

If using the *.cou* format, the **.EXTRACT** command also creates a *.ext* file when option **EXTFILE** is specified in the netlist, depending on the type of simulation and the command specification.

When extraction statements are combined with parameter sweeping statements (**.STEP**), Eldo automatically creates waveforms showing the extraction results versus the swept parameter. If using the *.cou* format, these **.EXTRACT** waveforms are created in a separate file with the *.ext* extension. The command-line switch **-couext** (**eldo -couext -i** <*netlist.cir*>) can be used to merge the extract waveforms into the *.cou* file, when possible. In this case, a single *.cou* file will contain the *raw* waveforms and also the extraction waveforms.

Related options

"AEX[=0|1|2]" on page 1010
"ALIGNEXT" on page 1011
"AUTOSTOP=0|1|2" on page 974
"AUTOSTOPMODULO=VAL" on page 975
"COMPEXUP" on page 951
"DEFRMSNTR" on page 999
"DUMP_EXTRACT=0|1" on page 999
"DUMP_MCINFO" on page 999
"ENGNOT" on page 995
"EXTCGS" on page 1000
"EXTERR" on page 952
"EXTFILE" on page 1000
"EXTMKSA" on page 1000
"EXTRACT_VECT_AXIS=INDEX|XAXIS" on page 1001
"NODEFRMSNTR" on page 1004
"NOEXTRACTCOMPLEX" on page 1004
"NUMDGT=INTEGER_VAL" on page 996

Parameters

To ensure that compatibility with previous versions of Eldo is maintained, it is preferable to select one of the optional parameters from the following list:

- **EXTRACT_INFO**

  should be replaced with one of the following parameters;

  AC

  Extraction during AC analysis.

  DC

  Extraction during DC analysis.

DCTRAN

Extraction after the DC analysis performed prior to a TRAN analysis.

DCAC

Extraction after the DC analysis performed prior to an AC analysis.

DCSWEEP

DCSWEEP extraction.

TRAN

Extraction during TRAN analysis.

NOISETRAN|NTR

Extraction during NOISETRAN analysis, see “.NOISETRAN” on page 747.

FOUR

Extract values from an FFT waveform.

DSP

Extract values from a DSP waveform, see “.DSP” on page 615.

SWEEP

Used in conjunction with **.STEP** to extract values from a curve (`EXTRACT = FUNCTION(SWEEP_parameter)`). It is also possible to specify the reference variable of a sweep. This reference variable can be a parameter or **TEMP** in the case of a **.TEMP** command.

SSTAC|SSTXF|SSTNOISE|SSTJITTER|TMODSST|FMODSST|FOURMODSST|TSST|FSST|SSTSTABIL

Please refer to the Display Command Syntax chapter of the *Eldo RF User’s Manual* for more information regarding these RF options.

OPT

Extraction during optimization analysis. Can only be used with wave type **WOPT**. This wave type is used for optimization analysis only, that is, it may only be used with **OPT**. For more information on wave type **WOPT**, see “WOPT” on page 1191.

- **LABEL**=NAME

  Label name of the extraction result. This can also be a string as shown below:

  ```
  .EXTRACT [EXTRACT_INFO] [LABEL=NAME] <expression>
  .EXTRACT [EXTRACT_INFO] [LABEL=" string "] <expression>
  ```

  If a `LABEL` is specified, then you will obtain `" LABEL "=result` inside the output file and not `<expression>=results`.

- **FILE**=FNAME

  Results will be dumped into the specified file. The values are also still written into the *.chi* file, that is, the same values and information are dumped in both files.

- **UNIT**=UNAME

  Defines the unit of the extract. It is used when plotting the extract wave during a sweep analysis. This could be useful when you want to compare an extracted wave to a reference because by default EZwave displays separate axis.

- **VECT**

  Extract vectors. By default, **.EXTRACT** returns the first value which matches the expression. If keyword **VECT** is set on the **.EXTRACT** statement, then all values will be returned. A waveform will be plotted in the *EXT* folder for extracts of type vector. This waveform will represent the value of the extract versus the index of the value.

  Specific values already extracted using the **.EXTRACT VECT** command can be used in a different **.EXTRACT** command. This is possible by identifying elements of a measurement specified with the **VECT** keyword with `[index]`. For example:

```
v1 1 0 pulse ( 0 5 0 1n 1n 10n 50n)
r1 1 0 1
v2 2 0 pwl ( 0 0 100n 10)
r2 2 0 1
.extract tran vect label = tito yval(v(2),xthres(v(1),2.5))
.extract tran meas(tito[2])
.tran 1n 100n
```

The first extraction will create a vector of five elements. The second extraction will extract item number 2 of the first extract.

Note

Extraction with **VECT** does not work when it is done in reverse. In the instance below,
.EXTRACT VECT xup(v(1),2.5,END,START)
the following message will be returned:
`VECT keyword cannot be used when extracting backwards`

- **CATVECT**

  Works in the same way as **VECT** but in addition all measurements corresponding to all analyses (**.STEP**/**.TEMP**) will be combined. This functionality is usually used in conjunction with the **CONTOUR** function in the **.PLOT** command. For more information on the **CONTOUR** function, see "CONTOUR" on page 798.

- **OPTIMIZER_INFO**

  Represents additional arguments for optimization. Note that these additional arguments (for example **GOAL**, **LBOUND**) have no effect when the **.OPTIMIZE** command is not specified in the netlist. For more information, please refer to ".EXTRACT and .MEAS" on page 1131.

- **MC_INFO**

  Represents additional arguments for Monte Carlo analysis, including:

Table 10-10. Monte Carlo Extract Outputs

LBOUND	UBOUND	MCMIN	MCAVG	MCSTD	MCVAR
MCSKEW	MCKURT	MCMOM	ICARLO	NBCARLO	

For more information, please refer to "Monte Carlo Output" on page 1215.

- **INTERP_MODE**

  Specifies the interpolation mode. Only applies to the yval() function. Given three simulation points: P1(x1,y1), P2(x2,y2) and P3(x3,y3); where x1, x2 and x3 represent the simulation variable (time, frequency, ...); and y1, y2, y3 represent the value of a waveform for x1, x2 and x3 respectively. Consider a search for value y = yval(<waveform>,x) which represents simulation point P(x,y), with P between P2 and P3. **INTERP_MODE** can take the following values:

  LINEAR: means a linear interpolation is performed between P2 and P3 (default).

  QUADRATIC: means a quadratic interpolation is performed between P1, P2 and P3.

  SAMPHOLD: means the value is constant between two points and equal to the previous point, (y = y2).

  HISTOGRAM: means the value is equal to the previous point until the mid-point is reached, and is equal to next point after it:
  if (x < (x2+x3)/2) y = y2;
  else y = y3;

  SPECTRAL: means the value is 0 if the point has not been simulated precisely.

- **TIME**=value

  Specifies a time for which an extract is active to extract information specific to one analysis. Only applies to AC and NOISE extracts when AC is performed during transient. For example:

```
.op 10n 30n 50n

.TRAN 1n 100n
.PLOT TRAN V(EP) V(S)

.AC DEC 10 1.0E3 1.0E9
.PLOT AC VDB(S)

.extract ac label=FOO1 yval(vdb(s),1g)
.extract ac label=FOO2 time=30n yval(vdb(s),1g)
.extract ac label=FOO3 yval(vdb(s),1g) time=50n
.extract ac label=FOO4 yval(vdb(s),1g) time=20n
```

For the extracts labelled FOO2 and FOO3, AC analysis information is extracted at the specified times. For the FOO4 extract, no extract is done, because no AC analysis is performed at 20 ns.

- `$MACRO`

  Instantiation of a macro previously defined using the **`.DEFMAC`** command. The macro name must be preceded by the **`$`** character.

- `FUNCTION`

  Pre-defined function. The functions listed under "FUNCTION" on page 645 are available. Many of these functions use the optional parameters listed in the table below:

Table 10-11. Extract Function Parameters

BEFORE=VAL	Causes the function to be performed only if TIME < val.
AFTER=VAL	Causes the function to be performed only if TIME > val.
OCCUR=VAL	Computes the function for the `VAL`th occurrence of the event.
VTH=VAL	Voltage defining a threshold or starting point.
VTHIN=VAL	Voltage defining a threshold or starting point.
VTHOUT=VAL	Voltage defining a threshold or starting point.
VH=VAL	Voltage defining a threshold or starting point.
VL=VAL	Voltage defining a threshold or starting point.
WAVE	Valid waveform name or keyword **`XAXIS`**. **`XAXIS`** extracts an x-axis value when a condition becomes true, that is, refers to the TIME if current simulation is transient, or frequency if current simulation is AC analysis. You can use wildcards in wave names.
MIN	Minimum value on x-axis or keyword **`START`**. **`START`** corresponds to the first value of the **`XAXIS`**, that is, the starting point of the simulation.
MAX	Maximum value on x-axis or keyword **`END`**. **`END`** refers to the last point of the simulation.

Functions can be used together with:

- Eldo operators such as the "Arithmetic Operators" on page 83 and "Boolean Operators" on page 83
- Output quantities (**`.PRINT`**/**`.PLOT`**), such as those listed in the tables under "Plot Specifications (OVN)" on page 799
- Pre-defined functions as listed here under "FUNCTION" on page 645
- Extract accessor functions

  Functions **`EXTRACT()`**, **`MEAS()`**, or **`INITIAL_EXTRACT()`** can be used in **`.EXTRACT`** statements to perform an "extract of extract." This can be useful if you want to extract the result of some extracts from a sweep analysis. If you want to use results of EXTRACT in another **.EXTRACT** command that applies to a single analysis, not to a sweep analysis, then specify the keyword **AC**, **TRAN**, or **DCSWEEP**, otherwise the value is not extracted. However, when there are no ambiguities, only one analysis being performed without **.STEP** or **.TEMP** commands, then the keyword is optional. A vector extract can also be used in an extract of extract, see extract of vector extract example.

EXTRACT(extract_label**)**

Extract accessor function used to access the value of a **.EXTRACT**, to perform an "extract of extract." For example:

```
.TEMP 10 20 30
.EXTRACT LABEL = T1 MAX(v(s))
.EXTRACT YVAL(EXTRACT(T1),20) ! extract T1 for T=20Celcius
```

Here, the second **.EXTRACT** is by default not evaluated after a single analysis, but only on completion of a SWEEP analysis; on completion of the three temperature analyses. See also a full extract of extract example.

MEAS(extract_label**)**

Alternative to the extract accessor function **EXTRACT**. Used to access the value of a **.EXTRACT**, to perform an "extract of extract."

INITIAL_EXTRACT(extract_label**)**

Extract accessor function used to access the value of a **.EXTRACT** inside subsequent **.ALTER** statements. A syntax error is issued if this accessor is used in a netlist without **.ALTER** statements or if the **.EXTRACT** value is not available for the nominal netlist. Otherwise it is replaced by the **.EXTRACT** value. For example:

```
.paramopt p1=(4,1,10)
V1 1 0 pwl(0 0 5n 0 6n 'p1' 10n 'p1' 11n 0)
R1 1 0 1
.tran 1n 15n
.extract tran label=FOO max(v(1))
.alter
.optimize
.objective tran label=MM max(v(1)) goal='0.9*INITIAL_EXTRACT(FOO)'
.end
```

Note

Transient extraction language functions and general purpose extraction language functions accept **EXTRACT()** and **MEAS()** keywords as parameters when they reference results from other **.EXTRACT** commands. However, transient functions are implicitly converted to general purpose functions in this case, and the memory and speed advantage of transient functions is lost.

In all cases below where ***VDD*** and ***VSS*** are used in function definitions, ***VDD*** represents the larger, and ***VSS*** the smaller voltage input found in the netlist at ***TIME***=0. These values are computed only once, even when a sweep is performed on ***VDD*** or ***VSS***.

For all function arguments requiring a value, a parameter name may be passed to the function, the parameter value being specified via the following command:

```
.PARAM PARAM_NAME=VAL
```

For wave arguments used in the functions below, either the name of a waveform created using the **.DEFWAVE** command or a waveform expression may be used. For example:

```
.extract max(W('V(s)-V(a)'))
```

which is equivalent to:

```
.defwave my_wave=V(s)-V(a)
.extract max(W(my_wave))
```

Two-port Noise Parameters

Any noisy two-ports can be represented by the equivalent noiseless two-port with the two equivalent noise sources (e_n and i_n) or the corresponding noise correlation matrix C^A. For more information on these parameters see "TWO_PORT_PARAM" on page 798.

It is also possible to use the center and radius of RF Gain/Noise circles in an extract command. The quantities are in the form:

```
<noise_circle>_R   !(real part)
<noise_circle>_I   !(imaginary part)
<noise_circle>_RAD !(radius)
```

where `<noise_circle>` can be any of the following: **GAC**, **GPC**, **LSC**, **SSC**, **NC**.

FUNCTION

There are two types of Extraction Language:

- Transient Extraction Language:
  Faster to execute, and consumes less memory. Transient extract language is based on the following functions:

Table 10-12. Transient Extraction Language Functions

D_WA	DTC	SLEWRATE	TCROSS	TINTEG
TPD	TPDUU	TPDUD	TPDDU	TPDDD
TPERIOD	TRISE	TFALL	VALAT	

- General Purpose Extraction Language:
  More expensive in terms of memory and CPU, but is much more general in the sense that arguments can be expressions. It is based on the following functions:

Table 10-13. General Extraction Language Functions

AVERAGE	COMPRESS	CROSSING	DCM	DISTO
EVAL	FAIL	FALLING	INTEG	KFACTOR
LOCAL_MAX	LOCAL_MIN	MAXGMVT	MAX	MEAN
MIN	MODPAR	OPMODE	PASS	PVAL
RISING	RMS	SLOPE	VDSATC	VTC
WFREQ	WINTEG	XCOMPRESS	XDOWN	XMAX
XMIN	XTHRES	XUP	XYCOND	YVAL

If `MIN` and `MAX` are not specified in a function call, information is returned when the `CONDITION` is true for the first time.

The x-axis values `MIN` and `MAX` may be replaced by the keywords **`START`** and **`END`** to specify the beginning and end of the simulation interval respectively. The x-axis value `MIN` may be made greater than the `MAX` value. This causes Eldo to look backwards when `CONDITION` is true for the first time. This is useful for extracting waveform settling time data.

AVERAGE

`AVERAGE(``WAVE [, MIN, MAX]`**`)`**

Returns the average value of the waveform `WAVE` in the x-axis range `MIN` to `MAX`. This is calculated as follows:

$$AVERAGE = \frac{1}{max - min} \times \int_{min}^{max} wavedx$$

COMPRESS

`COMPRESS``(WAVE, VALUE [, SLOPE])`

Extracts the Y-axis value of the wave at the point where the difference between the actual value of `wave` and the linear extrapolation of `wave` based on the computed slope value becomes greater than `value`.

`SLOPE` corresponds to the minimum slope which must be between the first two points of the wave. If the actual slope on the signal is not high enough, a message will be displayed. Default for this third argument is 0.99 if the `COMPRESS` is performed on a `SWEEP` extract. Otherwise, it is undefined.

See also XCOMPRESS and for further information see ".EXTRACT—Compression Point Values" on page 665.

CROSSING

CROSS[**ING**]**(**WAVE, VALUE[, MIN, MAX[,OCCURENCE]]**)**

Returns the x-axis value after the waveform `WAVE` has crossed `OCCURENCE` number of times the `VALUE` in the range `MIN` to `MAX`.

The **`CROSSING()`** function has two meanings when used inside **`XYCOND()`**, see "XYCOND" on page 660.

D_WA

D_WA(WAVE1, WAVE2 [, **AFTER**=VAL]**)**

Returns the distance between two waveforms. `D_WA` has the units of the waveforms, and is computed by dividing the area between the two waves by the x-axis range. `D_WA` may also be used in DCSWEEP to compare two characteristics of currents, for example:

```
.extract d_wa(v(a), v(b))
.extract d_wa(i(r1), id(m1))
```

DCM

DCM[**(**label_name**)**]

Returns the DC mismatch information for a specific `label_name` (see the .DCMISMATCH command). It extracts the value which is dumped in the *.chi* file, so you do not have to browse the *.chi* file to obtain that value. **`DCM`** can be specified without any argument (`label_name`) if there is no ambiguity about the name.

DELTA

DELTA(OVN[, **VGS**=VAL] [,**VDS**=VAL] [, **VBS**=VAL]**)**

Returns the difference between the value for the current run and the value calculated during the first run. Used with **`.STEP`** or **`.TEMP`** commands. `OVN` is any output quantity (**`.PRINT`**/**`.PLOT`**). **`VGS`**, **`VDS`** and **`VBS`** define the pin voltages used for device evaluation, and are required when device-related outputs are used, for example: IDS(M1), GM(X1.M2), VT(*). In the following example, Eldo computes the delta IDSAT quantities between the current run and the first run:

```
.extract label=delta_ids delta(IDS(*), VGS=3, VDS=3, VBS=0)
```

The **`DELTA`** function can also be used with reliability analysis to return the difference between aged and fresh values. See Delta Extract Results in the *Eldo UDRM User's Manual*.

DTC

DTC(WAVE [, **VTH**=VAL] [, **AFTER**=VAL] [, **BEFORE**=VAL])

Returns the duty-cycle of the waveform `WAVE` in transient analysis.

The duty cycle of the periodic waveform is the ratio of the "high" portion of the waveform to the length of the period.

The DTC extract is equivalent to the following three extracts if the first threshold is a rise time:

```
.extract tran label=period abs(xup(wave,vth,2) - xup(wave,vth,1))
.extract tran label=cycle abs(xdown(wave,vth,1) - xup(wave,vth,1))
.extract label=duty_cycle {extract(cycle)/extract(period)}
```

DISTO

```
DISTO(WAVE, FUND_FREQ, FMIN, FMAX)
```

Returns the total harmonic distortion of a signal. `FUND_FREQ` is the fundamental frequency and `FMIN` and `FMAX` specify the window in which you require the harmonic distortion to be calculated.

The harmonics inside the interval `[FMIN,FMAX]` are then computed as follows:

$$harmonic(i) = \left|\frac{A(i)}{A0}\right|$$

where:
A(i) = amplitudes of the multiples of the fundamental frequency
A0 = amplitude of the fundamental frequency

The Total Harmonic Distortion is given by the following:

$$\text{tot\_harm} = \sqrt{\sum harmonic(i)^2}$$

where the sum is computed over all multiples (≥2) of the fundamental frequency in the specified band. If these values are not identical to the sampled data values, then they are computed by interpolation.

The results of the harmonic distortion are given as a percentage (100×***tot_harm***).

```
.extract four disto(fourdb(fft_label),5meg,0,2.5e8)
.extract fsst disto(vm(node),75meg,10meg,600meg)
```

EVAL

```
E[VAL](INSTANCE, W|L|AD|AS|PD|PS|AREA)
E[VAL](INSTANCE, Weff|Leff|ADeff|ASeff|PDeff|PSeff|Geo|RDeff|RSeff)
E[VAL] (dipole_name)
```

Returns the value of the requested parameter for the circuit element `INSTANCE` or the `dipole_name`. The instance must be declared before the `.EXTRACT` command. For example:

```
.extract eval(m1, w)
```

The second syntax shows additional parameters which are only available for MOSFETs.

Specify option **EXTRACT_EVAL_FINAL** for **EVAL**(device_name,device_entity) to take into account the scale factor, option **SCALE**, if specified. See "EXTRACT_EVAL_FINAL" on page 1000.

FAIL

FAIL(label**)**

Provides information on whether an extraction passed or failed. Returns 1.0 if extract <label> was not successfully measured and 0.0 otherwise. label is the label name of the extraction result as specified in a previous extract statement. For example:

```
.extract tran label=UP500MV xup(v(IN),2.5)
.extract tran label=EX1 FAIL(UP500MV)
```

FALLING

FALLING(WAVE, VALUE[, MIN, MAX[,OCCURENCE]]**)**

Returns the x-axis value after the waveform WAVE has fallen OCCURENCE number of times below VALUE in the range MIN to MAX.

The **FALLING()** function has two meanings when used inside **XYCOND()**, see "XYCOND" on page 660.

INTEG

INTEG(WAVE [, MIN, MAX])

Returns the integral of the waveform WAVE in the x-axis range min to max. This is calculated as follows:

$$INTEG = \int_{min}^{max} wave dx$$

KFACTOR

Computes the stability factor for 2-ports. The keyword returns:

$$\frac{(1.0 - |S11|^2 - (|S22|^2 + |S11 \times S22 - S12 \times S21|^2))}{(2 \times |S12 \times S21|)}$$

S11, S22, and so on, are the S parameters. This is available with **AC** and **FSST** results.

LOCAL_MAX

LOCAL_MAX(WAVE [, MIN, MAX]**)**

Used without the **VECT** keyword, returns the first local maximum value of the waveform WAVE in the x-axis range MIN to MAX.

Used with the **VECT** keyword, returns all the local maxima of the waveform WAVE in the x-axis range MIN to MAX, and if option **EXTRACT_VECT_AXIS**=**XAXIS** is set (see

“EXTRACT_VECT_AXIS=INDEX|XAXIS” on page 1001) the resulting waveform matches the EZwave Local Max measurement, see Local Max in the *EZwave User’s and Reference Manual*.

LOCAL_MIN

LOCAL_MIN(WAVE [, MIN, MAX]**)**

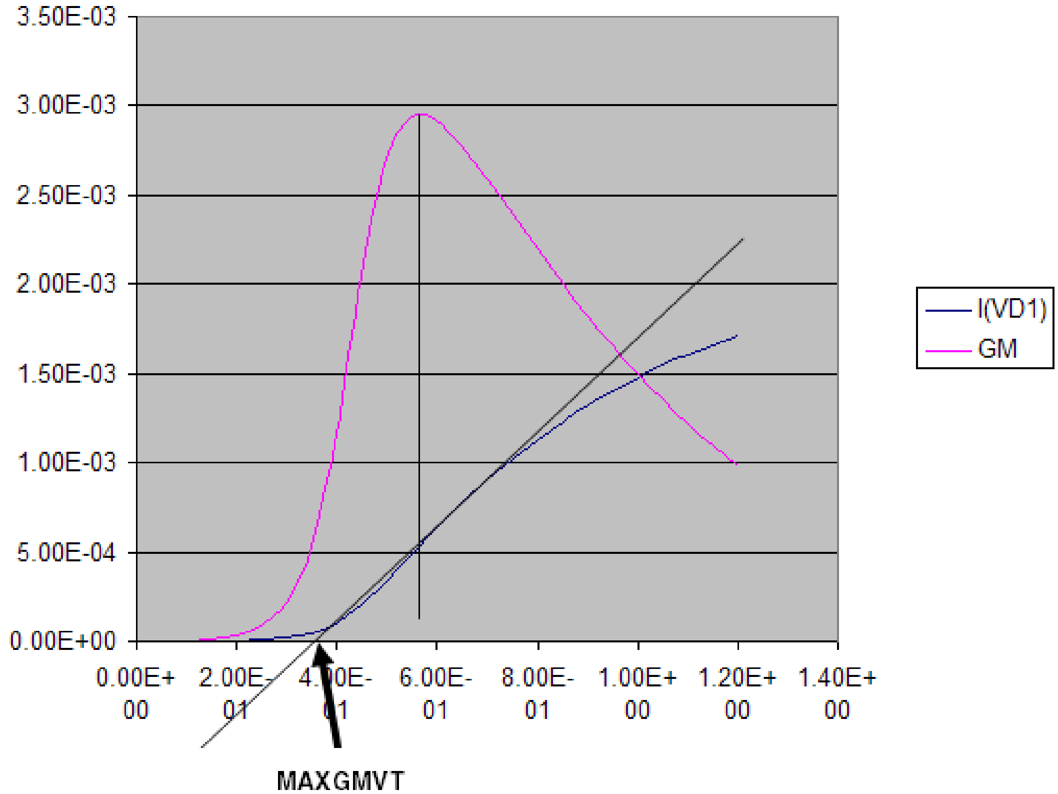
Used without the **VECT** keyword, returns the first local minimum value of the waveform WAVE in the x-axis range MIN to MAX.

Used with the **VECT** keyword, returns all the local minima of the waveform WAVE in the x-axis range MIN to MAX, and if option **EXTRACT_VECT_AXIS**=**XAXIS** is set (see “EXTRACT_VECT_AXIS=INDEX|XAXIS” on page 1001) the resulting waveform matches the EZwave Local Min measurement, see Local Min in the *EZwave User’s and Reference Manual*.

MAXGMVT

MAXGMVT(Mxx [, VDS, VGSMIN, VGSMAX]**)**

Returns the maximum transconductance threshold voltage for the MOS with instance name Mxx. The value is computed by sweeping the voltage across the gate and source of the MOS across a range defined by VGSMIN (minimum gate source voltage) and VGSMAX (maximum gate source voltage), and is the x-intercept of the maximum slope tangent line to an Id-Vg curve at a fixed Vds−Vds/2.



If `VDS` (drain source voltage), `VGSMIN` and `VGSMAX` are not specified then the following default values are internally calculated and used:

```
VDS = 0.1
VGSMIN = LV9(Mxx) - 0.5
VGSMAX = LV9(Mxx) + 0.5
```

Example using default values:

```
.extract dc label = MAX_GM_VT1 maxgmvt(m01)
```

Example when specifying the values:

```
.extract dc label = MAX_GM_VT2 MAXGMVT(M1, 0.1, 0, 2.0)
```

MAX

`MAX(``WAVE [, MIN, MAX]`**`)`**

Returns the maximum value of the waveform `WAVE` in the x-axis range `MIN` to `MAX`.

MEAN

`MEAN(``WAVE [, MIN, MAX]`**`)`**

Returns the mean value of the waveform `WAVE` in the x-axis range `MIN` to `MAX`. This is calculated as the sum of waveform values in [`MIN,MAX`] divided by the number of points between [`MIN,MAX`].

MIN

MIN(WAVE [, MIN, MAX]**)**

Returns the minimum value of the waveform `WAVE` in the x-axis range `MIN` to `MAX`.

MODPAR

MODPAR(MODNAME, PARAM_NAME**)**

Returns the value of the parameter, `PARAM_NAME`, of the model, `MODNAME`.

To extract the value of a parameter of a model defined inside a subcircuit:

- When no model parameters are an expression of an instance parameter, to extract a model parameter defined inside a subcircuit use the syntax modpar(subckt_name.model_name, parameter_name), for example:

  ```
  .extract dc modpar(toto.tutu, vt0)
  ```

- When some model parameters are an expression of an instance parameter, to reach a model parameter defined inside a subcircuit use the syntax modpar(hierarchical_instance_name.model_name, parameter_name), for example:

  ```
  .extract dc modpar(x1.tutu, vt0)
  ```

OPMODE

OPMODE(DEVICE_NAME**)**

Returns the DC working mode of the device model, `DEVICE_NAME`.

Eldo extracts the DC working mode of a device as a string instead of a real value. Strings returned are:

- for MOS devices:
  LINEAR, if $VGS > VT$ and $VDS < VDSAT$
  SATURATION, if $VGS > VT$ and $VDS > VDSAT$
  SUBTHRESHOLD, if $VGS < VT$

- for BJT devices:
  SATURATION, if $VBE > 0$ and $VBC > 0$
  ON, if $VBE > 0$ and $VBC < 0$
  OFF, if $VBE < 0$ and $VBC < 0$
  INVERSE, if $VBE < 0$ and $VBC > 0$

PASS

PASS(label**)**

Provides information on whether an extraction passed or failed. Returns 1.0 if extract `label` was successfully measured and 0.0 otherwise. `label` is the label name of the extraction result as specified in a previous extract statement. For example:

```
.extract tran label=UP500MV xup(v(IN),2.5)
.extract tran label=EX2 PASS(UP500MV)
```

PVAL

PVAL(parameter_name)

Returns the value of the requested parameter.

RISING

RISING(WAVE, VALUE[, MIN, MAX[,OCCURENCE]]**)**

Returns the x-axis value after the waveform WAVE has risen OCCURENCE number of times above VALUE in the range MIN to MAX.

The **RISING()** function has two meanings when used inside **XYCOND()**, see "XYCOND" on page 660.

RMS

RMS(WAVE [, MIN, MAX]**)**

Returns the root mean square value of the waveform WAVE in the x-axis range MIN to MAX.

This is calculated as follows:

$RMS = \sqrt{([\int_{min}^{max} wave^2]dx/(max - min))}$ for transient analysis and for Eldo RF time domain signals (TRAN, TSST)

$RMS = \sqrt{(\int_{min}^{max} wave^2)}$ for AC analysis and for Eldo RF frequency domain signals (AC, SST, NOISE, SSTNOISE, and so on)

For noise analysis, RMS calculations can not be performed in the same way as for other analyses, because of the specific nature of noise signals. RMS for noise analysis can be achieved with the following syntax:

```
.extract RMS(inoise)
.extract RMS(onoise)
.extract RMS(noise(Q1))
```

RMS(inoise) is calculated as:

$$RMS(inoise) = \sqrt{INTEG(INOISE \times INOISE)}$$

RMS(onoise) is calculated as:

$$RMS(onoise) = \sqrt{INTEG(ONOISE \times ONOISE)}$$

RMS(noise(object**))** is calculated as:

$$RMS(noise(object)) = \sqrt{INTEG(NOISE(OBJECT))}$$

SLEWRATE

```
SLEWRATE(WAVE [, VTH=VAL [, BEFORE=VAL] [, AFTER=VAL] [, OCCUR=VAL])
```

See the descriptions of these parameters under “FUNCTION” on page 643.

Returns the slope of the waveform `WAVE` at $v(wave) = vth$. Default is $vth = \frac{(VDD + VSS)}{2}$

SLOPE

```
SLOPE(WAVE, VTH [, MIN, MAX] [, n])
```

Returns the slope of the waveform `WAVE` at the nth occurrence of it crossing the y-axis value `VTH` in the x-axis range `MIN` to `MAX`.

TCROSS

```
TCROSS(WAVE [, VTH=VAL] [, OCCUR=VAL] [, AFTER=VAL] [, BEFORE=VAL])
```

Equivalent to the **`XTHRES`** function. **`TCROSS`** is part of the ‘Transient-extraction language’, while **`XTHRES`** is part of the ‘general-purpose extraction language’. The former mode is faster to execute, and consumes less memory, while the latter is more expensive in terms of memory and CPU, but is much more general in the sense that arguments can be expressions. The default value of **`VTH`** is calculated using the values of the voltage sources in the design. $VTH = vmin + 0.1 \times (vmax - vmin)$; where vmin is the lowest source value in the design and vmax is the highest value.

TINTEG

```
TINTEG(WAVE, [AFTER=VAL] [, BEFORE=VAL])
```

Equivalent to the **`INTEG`** function. **`TINTEG`** is part of the ‘Transient-extraction language’, while **`INTEG`** is part of the ‘general-purpose extraction language’. The former mode is faster to execute, and consumes less memory, while the latter is more expensive in terms of memory and CPU, but is much more general in the sense that arguments can be expressions.

TPD

```
TPD(WAVE1, WAVE2 [, VTH=VAL] [, VTHIN=VAL] [, VTHOUT=VAL]
+ [, BEFORE=VAL] [, AFTER=VAL] [, OCCUR=VAL])
```

See the descriptions of these parameters under “FUNCTION” on page 643.

Returns the propagation delay between `WAVE1` and `WAVE2`. The thresholds **`VTHIN`** and **`VTHOUT`** are defined for `WAVE1` and `WAVE2` respectively.

`VTHIN` is the threshold voltage to be used to detect transition on the first node given in the **`TPD`** command (for example **`TPD (`**`V(IN),V(OUT)`**`)`**), while **`VTHOUT`** is the threshold voltage for the second node (`V(OUT)` in this example). If **`VTHIN`** equals **`VTHOUT`**, it is possible to specify only **`VTH`**.

Default is $vth = \frac{(VDD + VSS)}{2}$.

If there is more than one transition, the average of the first and second is used. To measure specific signal transitions (for example, rising/falling on the input/output), the following functions may be used: TPDUU, TPDUD, TPDDU, and TPDDD.

If **OCCUR**=val is specified, Eldo returns the **TPD** at the val^th occurrence of the threshold crossing. The default value of **OCCUR** is 1. **OCCUR**=1 means Eldo returns the **TPD** value for the first occurrence of the threshold crossing, **OCCUR**=2 for the second occurrence, and so on. **OCCUR**=1 is between the first edge of WAVE1 and the next edge of WAVE2. **OCCUR**=2 is between the next edges of WAVE1 and WAVE2.

Alternatively, you can use **AFTER**=val and **BEFORE**=val in order to tell Eldo to return **TPD** values only if both threshold are crossed in the specified time interval.

TPD Examples

Assume V(IN) crosses **VTHIN** at 10n 50n and 100n,
assume V(OUT) crosses **VTHOUT** at 5n 75n and 110n:

```
TPD(V(IN),V(OUT),OCCUR=1)
```

would return 65n (75n - 10n), while:

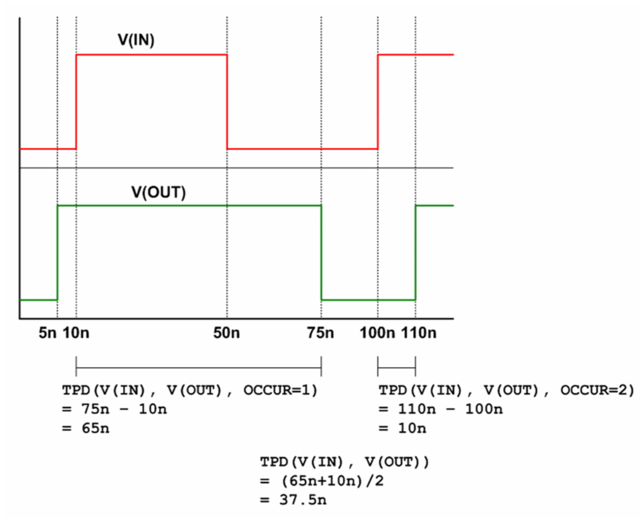
```
TPD(V(IN),V(OUT),OCCUR=2)
```

would return 10n (110n - 100n) (second occurrence).

However, **TPD**(V(IN),V(OUT)) would return 37.5n (average between the first **TPD** in one direction and the first **TPD** in the other direction).

Figure 10-4 illustrates this simple example of **TPD**.

Figure 10-4. TPD Example



TPDUU

```
TPDUU(WAVE1, WAVE2 [, VTH=VAL] [, VTHIN=VAL] [, VTHOUT=VAL]
+ [, BEFORE=VAL] [, AFTER=VAL] [, OCCUR=VAL])
```

`WAVE1` and `WAVE2` are both rising. For example:

```
.extract tran tpduu(v(in),v(out),vth=2.5,occur=3)
```

Eldo returns the propagation delay between v(in) and v(out) for the third occurrence of both waveforms rising above the 2.5 V threshold.

TPDUD

```
TPDUD((WAVE1, WAVE2 [, VTH=VAL] [, VTHIN=VAL] [, VTHOUT=VAL]
+ [, BEFORE=VAL] [, AFTER=VAL] [, OCCUR=VAL])
```

`WAVE1` is rising, `WAVE2` is falling. For example:

```
.extract tran tpdud(v(in),v(out),vth=2.5,occur=3)
```

Eldo returns the propagation delay between v(in) and v(out) for the third occurrence of waveform v(in) rising above the 2.5 V threshold and waveform v(out) falling below the same threshold.

TPDDU

```
TPDDU(WAVE1, WAVE2 [, VTH=VAL] [, VTHIN=VAL] [, VTHOUT=VAL]
+ [, BEFORE=VAL] [, AFTER=VAL] [, OCCUR=VAL])
```

`WAVE1` is falling and `WAVE2` is rising. For example:

```
.extract tran tpddu(v(in),v(out),vth=2.5,occur=3)
```

Eldo returns the propagation delay between v(in) and v(out) for the third occurrence of waveform v(in) falling below the 2.5 V threshold and waveform v(out) rising above the same threshold.

TPDDD

```
TPDDD(WAVE1, WAVE2 [, VTH=VAL] [, VTHIN=VAL] [, VTHOUT=VAL]
+ [, BEFORE=VAL] [, AFTER=VAL] [, OCCUR=VAL])
```

`WAVE1` is falling and `WAVE2` is falling. For example:

```
.extract tran tpddd(v(in),v(out),vth=2.5,occur=3)
```

Eldo returns the propagation delay between v(in) and v(out) for the third occurrence of both waveforms falling below the 2.5 V threshold.

TPERIOD

```
TPERIOD(WAVE [, VTH=VAL] [AFTER=VAL] [, BEFORE=VAL])
```

Returns the x-axis interval corresponding to the crossing of a y-axis value VTH by waveform WAVE in the x-axis range [AFTER, BEFORE]. The first crossing determines if the x-axis interval is measured between rising edges or falling edges.

The default value of VTH is calculated using the DC values of the voltage sources in the design. VTH = vmin + 0.1×(vmax – vmin) where vmin is the lowest source value in the design and vmax is the highest value.

TRISE

```
TRISE(WAVE [, VH=VAL] [, VL=VAL] [, BEFORE=VAL] [, AFTER=VAL]
+ [, OCCUR=VAL])
```

Returns the rise time of the next rising edge on `WAVE`.

TFALL

```
TFALL(WAVE [, VH=VAL] [, VL=VAL] [, BEFORE=VAL] [, AFTER=VAL]
+ [, OCCUR=VAL])
```

Returns the fall time of the next falling edge on `WAVE`.

See the descriptions of these parameters under “FUNCTION” on page 643. **VH** and **VL** are the high and low voltages defining the start and end points of the rise/fall. Defaults are:

$$vl = VSS + 0.1 \times (VDD - VSS)$$
$$vh = VSS + 0.9 \times (VDD - VSS)$$

VALAT

```
VALAT(WAVE, AT=VAL)
```

Returns the value of the waveform `WAVE` extracted at time `VAL`, for example:

```
.EXTRACT VALAT(v(s),AT=10n)
```

Returns the value of `v(s)` extracted at time 10ns.

VDSATC

```
VDSATC(Mxx [, IDS_NORM, VDS, VGSMIN, VGSMAX])
```

Returns the threshold voltage for the MOS with instance name Mxx. The value is computed by sweeping the voltage across the gate and source of the MOS across a range defined by VGSMIN and VGSMAX and is the VDSS for which the normalized drain source current IDS_NORM (Id×L/W) is equal to a given value at a fixed Vds.

If IDS_NORM, VDS, VGSMIN and VGSMAX are not specified then the following default values are internally calculated and used:

IDS_NORM = 600n
VDS = 0.1

VGSMIN = LV9(Mxx) - 0.5
VGSMAX = LV9(Mxx) + 0.5

Example using default values:

```
.extract dc label = VDSATC1 VDSATC(m01)
```

Example when specifying the values:

```
.extract dc label = VDSATC2 VDSATC(M1, 600e-9, 0.1, 0, 2.0)
```

VTC

VTC(Mxx [, IDS_NORM, VDS, VGSMIN, VGSMAX]**)**

Returns the threshold voltage for the MOS with instance name Mxx. The value is computed by sweeping the voltage across the gate and source of the MOS across a range defined by VGSMIN and VGSMAX and is the threshold voltage (VT) for which the normalized drain source current IDS_NORM (Id×L/W) is equal to a given value at a fixed Vds.

If IDS_NORM, VDS, VGSMIN and VGSMAX are not specified then the following default values are internally calculated and used: IDS_NORM = 600n, VDS = 0.1, VGSMIN = LV9(Mxx) - 0.5, VGSMAX = LV9(Mxx) + 0.5.

Example using default values:

```
.extract dc label = VTC1 VTC(m01)
```

Example when specifying the values:

```
.extract dc label = VTC2 VTC(M1, 600e-9, 0.1, 0, 2.0)
```

WFREQ

WFREQ(WAVE [, START, END]**)**

Returns the average frequency of the waveform `WAVE` between the times `START` and `END`, for example:

```
.EXTRACT WFREQ(V(OUTPUT), 10n, 100n)
```

Returns the average frequency of the voltage on node `OUTPUT` between `10n` and `100n`.

WINTEG

WINTEG(WAVE [, T]

Returns the time integral of the waveform `WAVE` from 0 to `T`. This is calculated as follows:

$$WINTEG = \int_0^t wave(u)du$$

When **`WINTEG`** is used in **`.DEFWAVE`** cards, the corresponding waves are dumped in *.meas* rather than in the binary output file.

XCOMPRESS

XCOMPRESS(WAVE, VALUE [, SLOPE]**)**

Extracts the X-axis value of the wave at the point where the difference between the actual value of wave and the linear extrapolation of wave based on the computed slope value becomes greater than value.

SLOPE corresponds to the minimum slope which must be between the first two points of the wave. If the actual slope on the signal is not high enough, a message will be displayed. Default for this 3rd argument is 0.99 if the XCOMPRESS is performed on a SWEEP extract. Otherwise, it is undefined.

Using the **COMPRESS**/**XCOMPRESS** function is especially useful in the case of parametrized simulations (**.STEP**). It works in the following way:

- compute the slope of wave on the first two points
- when the difference between the actual value of wave and the linear extrapolation of wave based on the slope value computed at step 1 becomes greater than value, then Eldo will return the value of the wave at that point (**COMPRESS** function), or the X-axis value at which the difference occurs (**XCOMPRESS** function).

See also COMPRESS and for further information see ".EXTRACT—Compression Point Values" on page 665.

XDOWN

XDOWN(WAVE, VTH [, MIN, MAX] [, n]**)**

Returns the x-axis value of the waveform WAVE at the nth occurrence of it falling below a y-axis value VTH in the x-axis range MIN to MAX.

XMAX

XMAX(WAVE [, MIN, MAX]**)**

Returns the value of the x-axis when MAX value is reached, for example:

```
.EXTRACT TRAN XMAX (v(out))
```

Returns the time at which v(out) reaches its maximum value.

XMIN

XMIN(WAVE [, MIN, MAX]**)**

Returns the value of the x-axis when MIN value is reached.

XTHRES

XTHRES(WAVE, VTH [, MIN, MAX] [, n]**)**

Returns the x-axis value of the waveform `WAVE` at the nth occurrence of it crossing a y-axis value `VTH` in the x-axis range `MIN` to `MAX`.

XUP

XUP(WAVE, VTH [, MIN, MAX] [, n]**)**

Returns the x-axis value of the waveform `WAVE` at the nth occurrence of it rising above a y-axis value `VTH` in the x-axis range `MIN` to `MAX`, for example:

```
.EXTRACT XUP(V(S),2.5,100n,300n,5)
```

Returns the x-axis value when `V(S)` rises above 2.5 on the y-axis for the 5th time between 100n and 300n.

Note

For **SLOPE**, **XDOWN**, **XTHRES**, and **XUP** functions: occurrence is always assumed to be 1 if the search occurs in the reverse direction, for example:

XUP(V(s),2.5,end,start)
XUP(V(s),2.5,end,start,5)

Both will return the last x-axis value when `V(S)` crosses above 2.5 on the y-axis.

XYCOND

XYCOND(WAVE, CONDITION [, MIN, MAX]**)**
XYCOND(WAVE, WAVE2=vth**)**

Returns the value of the waveform `WAVE` when the expression represented by `CONDITION` is true for the first time, evaluated between the limits `MIN` to `MAX`. Parameter `WAVE` may be a valid waveform name or the keyword `XAXIS` which extracts an x-axis value when a condition becomes true. The parameter `CONDITION` may contain any of the available arithmetic expressions.

Table 10-14. XYCOND Arithmetic Expressions

==	Equal
!=	Not equal
||	Or
&&	And
<	Less than
>	Greater than

See "Arithmetic Functions" on page 78.

The **RISING()**, **FALLING()**, and **CROSSING()** functions have two meanings when used inside **XYCOND()**. Specified inside the condition argument, they are boolean functions which return 1 if the signal crossed the threshold, and 0 otherwise. Specified inside the start and end arguments (MIN, MAX), they are equivalent to **XUP()**, **XDOWN()**, and **XTHRES()** respectively.

YVAL

YVAL(WAVE, X_VALUE**)**

Returns the y-axis value of waveform WAVE when an x-axis value X_VALUE is reached. Complex results information instead of real values are enabled by default. For example, `.extract ac label=EX1 yval(v(out),10k)` would deliver the result:

```
* EX1 = 10.011 , -1.8
```

The general form is: real_part, imaginary_part.

Setting option **NOEXTRACTCOMPLEX** (see "NOEXTRACTCOMPLEX" on page 1004) disables this complex result information.

Keyword INTERP_MODE can be used to specify the interpolation mode. It applies only to the yval() function.

When the vector mode extract is specified (keyword VECT) then the x-axis value is automatically incremented even when explicitly specified. For example:

```
.extract ... vect yval(v(a), 10n)
```

generates the v(a) value at every 10ns interval.

Examples

This example extracts the time when the voltage on node 1 rises to a value of 2.5V between the range 0 and 100ns and lists the results to the *.chi* file:

```
.extract xup(v(1), 2.5, 0, 100n)
```

The example below extracts the maximum value of the user defined waveform power_vdd and lists the results to the ASCII output file:

```
.extract max(w(power_vdd))
```

The example below extracts the average value of the power waveform of voltage source V1 (equivalent to **.CONSO** command) and lists the results to the ASCII output file:

```
.extract tran average(pow(V1))
```

The example below extracts the value of the power waveform of voltage source V1 at T=0 and lists the results to the ASCII output file:

```
.extract tran pow(V1)
```

The example below extracts the first occurrence of the phase of the voltage on node `s` when the magnitude of the voltage on node `s` is less than 0dB in the range of the simulation interval and lists the results to the ASCII output file:

```
.extract xycond(vp(s), vdb(s)<0.0, start, end)
```

The next, more complex, example extracts the delay between the voltages on nodes `1` and `2` of the circuit and lists the results in the *.chi* file:

```
.extract xup(v(1), 2.5, 0, 100n) –
+ xdown(v(2), 2.5, 0, 100n)
```

The example below defines a macro called `phmag` with arguments `a` and `b`. This macro is then instantiated via the **`.extract`** command with the arguments `a` and `b` being replaced by the phase voltage **`vp(`**`s`**`)`** and magnitude **`vdb(`**`s`**`)`** on node `s` respectively. The results are listed in the ASCII output file. The `phmag` macro uses the pre-defined functions **`xycond`** and **`yval`**:

```
.defmac phmag(a, b) = xycond(a, b<0.0) – yval(a, 0.001)
...
.extract $phmag(vp(s), vdb(s))
```

The example below extracts the value of waveform `v(1)` when waveform `v(2)` reaches 3 for the first time.

```
.extract tran xycond(v(1),v(2)==3)
```

It is also possible to extract values from a FFT waveform. Eldo will print out the value (in dB) at 5 Meg for the FFT done on `v(A)`:

```
.FOUR LABEL = t1 V(a)
.EXTRACT FOUR YVAL(FOURDB(t1),5MEG)
```

In the example below, 10 simulations will be made with `P1` varying from 1 to 10. There will be an extracted value for each of the 10 values of `P1`. The *.ext* file contains the information ***EXTRACTED_VALUES = f(stepped_value)***; this file can be read by EZwave:

```
.STEP P1 1 10 1
.EXTRACT MIN(V(OUT))
```

In this next example, eleven analyses are performed for `P1` varying from 1p to 10p. After each analysis, `MAX(VDB(S))` is extracted; at that step, the second **`.EXTRACT`** command is ignored. Once the 11 simulations are complete, Eldo will interpret the **`.EXTRACT SWEEP`**... commands, and will then extract `toto`, that is, `MAX(VDB(S))` when `P1` was 5p:

```
.PARAM p1 = ..
C1 S 0 p1
.STEP PARAM P1 1p 10p 1p
.EXTRACT LABEL=toto MAX(VDB(S))
.EXTRACT SWEEP YVAL(MEAS(toto),5p)
```

The next example shows how to use the results of EXTRACT in another **`.EXTRACT`** command that applies to a single analysis, with the keyword **`TRAN`** specified. When there are no ambiguities, that is, only one analysis being performed, without **.STEP** or **.TEMP** commands, then the keyword is optional:

```
.TRAN 1n 10n
.TEMP 10 20 30
.EXTRACT LABEL = T1 MAX(v(s))
.EXTRACT LABEL = T2 MIN(v(a))
.EXTRACT TRAN EXTRACT(T1)-EXTRACT(T2) ! keyword TRAN expected
```

The following is an example of the results of an **.EXTRACT** command dumped into a specified file *file.aex*:

```
EXTRACT for AC ANALYSIS
 TEMPERATURE = 2.700000e+01 Celcius
  *YVAL(VDB(NET4),100K) = -1.01E+02
EXTRACT for TRANSIENT ANALYSIS
 TEMPERATURE = 2.700000e+01 Celcius
  *V(NET4) = 1.93E-09 Volts
  *YVAL(V(NET4),50N) = 1.40E+00
```

The following example shows how the **VECT** keyword can be used to return all the values which match the expression. By default, **.EXTRACT** returns the first value which matches the expression.

```
v1 1 0 pulse(0 5 0 1n 1n 8n 20n)
r1 1 0 1
.extract vect xup(v(1),2.5)
.meas vect tran memes trig at 0 targ v(1) val=2.5 rise=1
.tran 1n 100n
.end
```

The following examples show how gain, phase margin, and gain margin can be extracted.

```
.EXTRACT label=gain max(W('v(output)/v(input)'))
.EXTRACT ac label=phmargin
+ {180 - yval(vp(VO),1m) + xycond(vp(VO),vdb(VO)<0)}
.EXTRACT ac label=gmargin
+ xycond(vdb(VO), 180+vp(VO)<yval(vp(VO),1m))
```

It is possible to specify the reference variable of a sweep in the **SWEEP** keyword. This reference variable can be a parameter or **TEMP** in the case of a **.TEMP** command. The example below shows two nested sweeps, and an extract. The *.wdb* file will contain five waveforms for EXTRACT(T1) with the x-axis being the phase value and one waveform with the x-axis being FREQU value. A new waveform will be generated which is the value of the extract as a function of the second sweep parameter. An extract on this waveform is performed, then the value of the second extract as a function of the first swept parameter is obtained.

```
* nested sweep extract
.param frequ=1k phase=0
v1 1 0 sin (0.5 0.5 frequ 0 0 phase)
r1 1 0 1k
.tran 0 '2/frequ'
.step param frequ 0.8 1.2 INCR 0.1
.step param phase list -180 -90 0 90 180
.extract tran label=t1 xthres(v(1), 0.5) unit=Time ! versus Phase
.EXTRACT SWEEP(frequ) label=t2 max(meas(t1)) unit=Time ! versus Freq
```

You can access a single value of a vector extract. For example, if the following command line:

```
.EXTRACT tran vect label=fall xdown(v(ep),0.8)
```

generates the following output:

```
FALL[1] =  4.1200E-08
FALL[2] =  1.2120E-07
FALL[3] =  2.0120E-07
FALL[4] =  2.8120E-07
```

Then it is possible to use the second value in another EXTRACT expression using:

```
EXTRACT(FALL[2])
```

A full extract of extract example can be found in the software delivery tree: *$MGC_AMS_HOME/examples/eldo/extract_of_extract.cir*. The netlist is shown below:

```
* Extract Exercice

fns in out 1, 1 1e-7 8e-14
.param vhi=1
vin in 0 dc 0 ac 1 pulse 0 vhi 1u 1n 1n 5u 20u

*** AC Extraction ***
.tran 1u 10u
.plot tran v(in) v(out)
.probe tran v(out)

*** TRAN Extraction ***
.extract tran label=maxwave max(v(out))
.extract tran label=maxofmax max(extract(maxwave))

.option eps=1e-7
.step param vhi 5 1 1
.end
```

The extract labelled MAXWAVE is performed for each value of parameter VHI. EXTRACT(MAXWAVE) represents a waveform with VHI values for the x-axis and MAXWAVE values for the y-axis. Then the extract labelled MAXOFMAX is performed when all values of MAXWAVE are available: once the sweep is complete.

The following example shows another extract of extract scenario, where the first **`.EXTRACT`** returns a vector. For example, the statement below extracts the four time points at which v(clk) drops to 0.4V:

```
.extract vect tran label=clk_fall xdown(v(clk), 0.4)
```

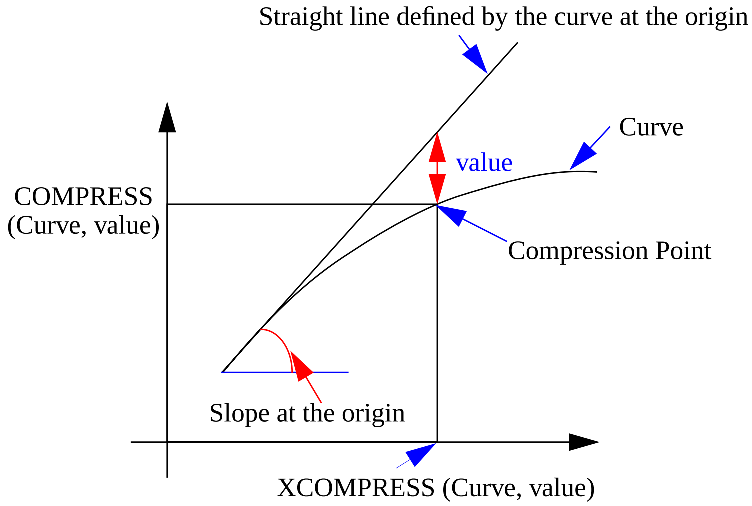
Then to extract the value of v(TOP) at the four time points already calculated for clk_fall above, use:

```
.extract tran label=data_vals yval(v(top), meas(clk_fall[]))
```

.EXTRACT—Compression Point Values

The **`COMPRESS`**/**`XCOMPRESS`** functions extract the co-ordinates of a compression point in the curve specified in the first parameter of the function. **`XCOMPRESS`** provides the X-value and **`COMPRESS`** the Y-value.

This compression point is the point where the curve is `value` below the straight line defined by the slope of the curve at the origin. (`value` is the second parameter of the function.)



In the illustration above, the curve is `meas(POUTDBm)`, and the compression `(value)` is 1.0 dBm.

Caution

When these functions are used to extract compression points, the swept parameter (X-axis) and the curve (Y-axis) must be in dB (or dBm).

Example

```
* 1dB compression point
.param fund=900Meg
.extract fsst label=POUTdBm YVAL(PdBm(RL), fund)
.extract sweep xcompress(meas(POUTdBm), 1.0)
.extract sweep compress(meas(POUTdBm), 1.0)
```

.EXTRACT—Used with Monte Carlo Analysis

At the end of a Monte Carlo simulation, Eldo prints in the *.chi* file a histogram for each **`.EXTRACT`** command. The example below shows the *.chi* file entry for a Monte Carlo analysis with the following **`.EXTRACT`** command:

```
.EXTRACT tran label=tpd tpdud(v(in), v(out))

Distribution of TPD
                  Range [ 9.8405E-10  1.7374E-09]
                  Nominal value:  1.3124E-09
                  Average value:  1.2709E-09
```

```
                     Standard Deviation:  1.9007E-10
                     Standard Deviation based on nominal run:  1.3124E+00

   [ 984.00000P      1.06060N  ] NB = 2  FREQ = 1.00e+01% |*****
   [ 1.06060N        1.13720N  ] NB = 3  FREQ = 1.50e+01% |*******
   [ 1.13720N        1.21380N  ] NB = 5  FREQ = 2.50e+01% |************
   [ 1.21380N        1.29040N  ] NB = 1  FREQ = 5.00e+00% |**
   [ 1.29040N        1.36700N  ] NB = 4  FREQ = 2.00e+01% |**********
   [ 1.36700N        1.44360N  ] NB = 1  FREQ = 5.00e+00% |**
   [ 1.44360N        1.52020N  ] NB = 2  FREQ = 1.00e+01% |*****
   [ 1.52020N        1.59680N  ] NB = 1  FREQ = 5.00e+00% |**
   [ 1.59680N        1.67340N  ] NB = 0  FREQ = 0.00e+00% |
   [ 1.67340N        1.75000N  ] NB = 1  FREQ = 5.00e+00% |**
```

To produce a histogram such as the one shown above, a Monte Carlo analysis (**.MC**) with at least nine runs must be performed.

Note

The histogram is also output in the binary output file and can be displayed with EZwave.

For more information about Monte Carlo Analysis, see “.MC” on page 706 and the Monte Carlo Analysis chapter, and for output extraction see “Monte Carlo Output” on page 1215.

.FFILE

S, Y, Z Parameter Output File Specification

```
.FFILE S|Z|Y|G|H|T|A [SINGLELINE] FILENAME [HZ|KHZ|MHZ|GHZ] [RI|MA|DB]
```

For more information, see "Output File Specification" on page 1045 and "Touchstone Data Format" on page 1058.

Parameters

- **S**

  Specifies S (Scattering) frequency parameters tabulation.

- **Y**

  Specifies Y (Admittance) frequency parameters tabulation.

- **Z**

  Specifies Z (Impedance) frequency parameters tabulation.

- **G**

  Specifies G (Hybrid-G) matrix parameters tabulation.

- **H**

  Specifies H (Hybrid-H) matrix parameters tabulation.

- **T**

  Specifies T (transfer scattering) matrix parameters tabulation.

- **A**

  Specifies A (chain or ABCD) matrix parameters tabulation.

- FILENAME

  Name of the file where the S, Y or Z parameters will be stored.

- **SINGLELINE**

  This enables the user to obtain the S-parameter file in single line format as shown below.

  ```
  Freq S11 S21 S12 S22
  ```

- **HZ**

  Specifies the units to be Hz. This is the default.

- **KHZ**

  Specifies the units to be kHz.

- **MHZ**

  Specifies the units to be MHz.

- **`GHZ`**

  Specifies the units to be GHz.

- **`RI`**

  Specifies Real Imaginary storage format. This is the default.

- **`MA`**

  Specifies Magnitude Angle storage format.

- **`DB`**

  `MA` with magnitude in dB.

Two-port noise parameters NFMIN_MAG, GAMMA_OPT_MAG, PHI_OPT and RNEQ are automatically written to the specified output file when a **`.NOISE`** command is specified in the netlist and the circuit to be analyzed is a two-port circuit.

Note

`.FFILE` can be used only with AC analysis, not SST.

Examples

For examples, see "Output File Specification" on page 1045.

.FILTER

Filter Data Driven Simulations

```
.FILTER AC|DC|TRAN|MODSST|SST|ALL [condition]
.FILTER EXTRACT [LABEL=label] [condition]
.FILTER DATA DATA_IN=input_data DATA_OUT=output_data [condition]
```

This command provides filtering capability for simulations, extractions, or to create subsets of **.DATA** statements. The filter is a logical expression, using the **.PARAM** syntax. Expressions can be entered directly or as a reference to a **.PARAM**. The filtering expression can involve the parameters used in a **.DATA** block in case of data driven simulations, usual parameters defined by **.PARAM**, and predefined keywords which identify an internal state, such as a Monte Carlo run index. If no condition is specified, all entries are taken into account.

Note

.FILTER is compatible with **.OPTIMIZE**, but is not supported in ADMS.

The use of **.FILTER** allows to define the objective waveform as a subset of the global **.DATA** which may have been automatically generated by another tool.

Reserved keywords for internal quantities are:

ISTEP	for the index of the current step (first index is 0)
ICARLO	for the index of the current Monte-Carlo run (first index is 0)
IALTER	for the index of the current **.ALTER** (first index is 0)
ITEMP	for the index of the current temperature value (first index is 1)
NBSTEP	for the total number of values generated by **.STEP** commands
NBCARLO	for the total number of Monte-Carlo runs
NBALTER	for the total number of **.ALTER**
NBTEMP	for the total number of temperature values

Parameters

- **AC|DC|TRAN|MODSST|SST|ALL**

  Specifies the analysis type.

- condition

  For the filter of simulation syntax, if condition is false, then the corresponding analysis is bypassed. This is especially useful in the case of multiple simulations triggered by the presence of **.DATA**, and if only a few points of the data are of interest.

For the filter of extractions syntax, if `condition` is false, then the corresponding **.EXTRACT** or **.MEAS** is bypassed. If no label is given, all **.EXTRACT** and **.MEAS** commands are bypassed when condition is false.

For the filter of data syntax, `condition` should be a logical expression using only entries of the input **.DATA** block. Each set of values will be evaluated. If `condition` is false, the set of values will not appear in the output **.DATA** block. This is especially useful in case of optimization.

- **LABEL**=label

  Specifies the label of a **.EXTRACT** or **.MEAS**.

- **DATA_IN**=input_data

  Specifies the name of a **.DATA** statement.

- **DATA_OUT**=output_data

  Specifies the name of the **.DATA** statement which will be created from the filtering of **.DATA** input_data.

Examples

```
.TRAN 0 100n sweep data = data_block
.FILTER all ptemp==27 && pvdd==3
.DATA data_block
+ ptemp pother pvdd res1 res2
+ 27 456  3  1e-5 1e8
+ 27 678  3.2  1e-5 1e8
+ 27 321  3.2  1e-5 1e8
+ 27 12    3  1e-5 1e8
+ 27 458  3.2  1e-5 1e8
.enddata

v1 1 0 pvdd
r1 1 0 1
.extract tran yval(v(1),0)
.end
```

In this example, simulation will be performed where ptemp=27 and pvdd=3. This corresponds to the fourth entry of the **.DATA** block. All other simulations will be bypassed.

```
.STEP PARAM p1 1 10 1
.EXTRACT tran label=FOO max(v(1))
.FILTER EXTRACT LABEL=FOO {p1 >= 3}
```

This example filters extractions, when p1>=3 the corresponding **.EXTRACT** is taken into account.

```
.option OPSELDO_OUTER
.TEMP -10 27 50
.TRAN 1n 100n
.FILTER data data_in=IIN data_out=OON {abs(ptemp-temper)<1e-5}
.objective tran label=fit_out V(S) GOAL=OON(VS)
```

```
.data IIN x ptemp vs
+  10n -10 5.142e-5
...
+  20n -10 1.023e-1
+  30n -10 7.175e-1
+  40n -10 1.025
+  50n -10 1.018
+  60n -10 9.710e-1
+  70n -10 3.717e-1
+  80n -10 -5.221e-2
+  90n -10 1.674e-3
+ 100n -10 2.031e-3

+  10n  27 5.955e-5
+  20n  27 1.463e-1
+  30n  27 9.377e-1
+  40n  27 1.051
+  50n  27 9.956e-1
+  60n  27 9.556e-1
+  70n  27 1.503e-1
+  80n  27 -3.140e-2
+  90n  27 5.975e-3
+ 100n  27 -1.336e-4

+  10n  50 6.450e-5
+  20n  50 6.155e-2
+  30n  50 4.499e-1
+  40n  50 7.884e-1
+  50n  50 9.671e-1
+  60n  50 1.019
+  70n  50 7.394e-1
+  80n  50 3.579e-1
+  90n  50 6.591e-2
+ 100n  50 -4.888e-2
.enddata
```

This example shows the filter of data. In this example three optimizations are required, one for each temperature.

.FORCE

Initial Transient Analysis Conditions

```
.FORCE [NODE] {node_name value}
+ [IND|OBJ] {object_name value}
```

Forces one or more nodes to specified voltage(s) with respect to ground for the initial transient solution. This is similar to the .IC command, values will be used only for the DC performed prior to TRAN analysis. Both commands are used to give values replacing the DC solution. With the **.FORCE** syntax, initial values of inductors current can be given while with **.IC** only node voltages can be initialized including nodes inside subcircuits.

If the **UIC** parameter is also present (in the **.TRAN** command) no DC analysis is performed and the voltages are initialized as defined in the **.FORCE** command. All other voltages on nodes not initialized in the **.FORCE** command are determined by Eldo itself. During subsequent analysis (transient), the node voltages are freed of their initial values, and may therefore assume different values.

Parameters

- `node_name`

  Node name.

- `value`

  Value at node.

- **IND|OBJ**

  Inductor or object. For objects, only voltage source components can be specified to set the current on.

- `object_name`

  Inductor or voltage source component.

Example

```
V1 1 0 pwl(0 0 10n 5v)
R1 1 2 1k
C1 3 0 100p
L1 2 3 1u
.force ind L1 10mA
.tran 1n 0.1u uic
.plot tran  i(L1)
.plot tran v(3)
.end
```

.FOUR

FFT Select Waveform

```
.FOUR LABEL=label_name waveform_name [optfour_parameters]
```

Selects the waveform on which you require FFT to be done. This command also accepts signals, for example `.FOUR sg(ycore->A(0))`.

Options for the FFT post-processor can be specified inside this command or through the **`.OPTFOUR`** command, see “.OPTFOUR” on page 764.

Parameters

- `label_name`

  Reference name used by Eldo to select which wave has to be plotted (**`.PLOT`**) or on which FFT wave extraction of useful quantities has to be done (**`.EXTRACT`**). `label_name` is required.

- `waveform_name`

  Any regular Eldo waveform name: there can be any number of **`.FOUR`** commands specified in the *.cir* file.

- `optfour_parameters`

  `.OPTFOUR` command parameters can be specified here. This allows calculation of multiple FFTs in a single simulation. If no additional arguments are provided, the **`.FOUR`** command uses the default setting of **`.OPTFOUR`** parameters.

Examples

```
.FOUR LABEL = t1 V(a)
.FOUR LABEL = t2 W('V(a) - v(b)/2.0')

.FOUR LABEL=foo1 v(ofdm).h(1)
+ tstart=modsst_fft_tstart1 tstop=modsst_fft_tstop1
+ nbpt=48 normalized=1 interpolate=0
+ display_input=1
```

The last example shows the specification of **`.OPTFOUR`** command parameters inside the **`.FOUR`** command. This allows calculation of multiple FFTs in a single simulation.

Display and .EXTRACT of FFT results

```
[.PLOT|.PRINT|.PROBE ] FOUR FOURxx(label_name)
.EXTRACT FOUR <expression>
```

`xx` stands for **`DB`**, **`R`**, **`I`**, **`P`**, **`M`**.

Examples

```
.FOUR LABEL = t1 V(a)
.PLOT FOUR FOURDB(t1)
```

It is also possible to extract values from a FFT waveform:

```
.FOUR LABEL = t1 V(a)
.EXTRACT FOUR YVAL(FOURDB(t1),5MEG)
```

Eldo will print out the value (in dB) at 5 Meg for the FFT done on `v(A)`.

.FUNC

User Defined Function

```
.FUNC P(a,b,...) EXPR
```

This command is used to define functions used in expressions. They are flexible and useful when there are several similar sub-expressions in a circuit file.

This command provides compatibility with PSpice. It is equivalent to the .PARAM user-defined function specification.

Parameters

- `P(a,b) EXPR`

  Specifies a user-defined function in order to define a parameter using an expression. The parameter may then be called when required.

Example

```
.func P(a,b) expression
R1 1 2 'P(a,b)'
```

.GLOBAL

Global Node Allocation

```
.GLOBAL NN {NN}
```

Declare global node(s), making them known throughout a circuit without having to declare them in each subcircuit. The number of such nodes is unlimited. **.GLOBAL** is active when placed anywhere in the netlist. This applies to any launch mode.

By default, nodes which appear in subcircuit definitions have priority over the nodes defined with **.GLOBAL** statements. It is exactly the opposite when option **DSCGLOB=GLOBAL** is specified (must be at the beginning of the netlist): nodes defined with **.GLOBAL** statements take priority over nodes which appear in subcircuit definitions.

Parameters

- NN

  Name(s) of the node(s) to be declared as global.

Example

```
.global vdd vss
```

Specifies vdd and vss as global nodes throughout the circuit.

```
.global vdd mid

.subckt div1 mid
r1 vdd mid 1k
r2 mid 0 1k
.ends

X1 net1 div1
r1 net1 0 1k
```

By default, Eldo will connect X1.r1 and X1.r2 to node net1. If option **DSCGLOB=GLOBAL** is set X1.r1 and X1.r2 will be connected to global node mid.

See also "DSCGLOB Option Example" on page 903.

Related Option

DSCGLOB, see "DSCGLOB=X|GLOBAL" on page 976.

.GUESS

Initial DC Analysis Conditions

```
.GUESS V(NN)=VAL [SUBCKT=subckt_name] {V(NN)=VAL [SUBCKT=subckt_name]}
```

Helps to calculate the DC operating point by setting voltage values at selected nodes for the first iteration of a DC operating point calculation.

This command differs from the **.NODESET** command in so far as when using **.GUESS** the node voltages are only fixed for the first iteration of a DC operating point calculation, whereas when using **.NODESET** the node voltages are fixed for the duration of the first DC operating point calculation.

This command is useful when the approximate whereabouts of the DC operating point is known, enabling the simulator to converge more quickly.

Note

By default, the first **.GUESS** specification has precedence over subsequent **.GUESS** specifications. Setting **.OPTION LICN**, the last **.GUESS** specification will have precedence.

See "LICN" on page 979 for further information.

Parameters

- **V(**NN**)**=VAL

  Voltage at node NN in volts.

- **SUBCKT**=subckt_name

  If specified it will fix the voltage of the preceding node in all instances of the subcircuit subckt_name.

Example

```
.guess v(n4)=6v v(n5)=2v v(n6)=-5v
```

Specifies that for the first iteration of a DC operating point calculation the voltages at the nodes n4, n5 and n6 be set to 6V, 2V and –5V respectively.

```
.guess v(2)=3v SUBCKT=sub1 v(4)=-2v SUBCKT=sub2
```

Specifies that for the first iteration of a DC operating point calculation the voltages at node 2 of subcircuit sub1 and node 4 of subcircuit sub2 will be set to 3 and –2V respectively.

.HDL

Compile and Load Verilog-A Modules

```
.HDL FILE=filename [MODULE=module_name] [ALIAS=alias_name]
```

Compiles and loads Verilog-A modules in Eldo. The modules compiled with **.HDL** are instantiated in the same manner as Eldo subcircuits with names beginning with the “x” character. The module parameters can be provided on a **.MODEL** statement or on the module instance.

If only `module_name` is specified then `module_name` will be compiled from the file and loaded in the simulator. If in addition `alias_name` is specified then any reference to `module_name` must be done using `alias_name` instead of `module_name`.

In addition to the **.HDL** command, three related command line flags and three related options are available: `-hdlpath`, `-spmodel`, `-vamodel` and **SPMODEL**, **VAMODEL**, **VAOPTS**.

Refer to the *Eldo Verilog-A User’s Manual* for further information.

Parameters

- **FILE**=filename

  Verilog-A file.

- **MODULE**=module_name

  Module name. Optional. If specified, only that module is loaded from the specified Verilog-A file.

- **ALIAS**=alias_name

  Specifies an alias instead of the module name defined in the Verilog-A file. Optional. This could be useful if you want to load modules of the same name from different source files.

Examples

```
.hdl mos.va
.model myjfet jfet lambda=0.01
x1 d g s myjfet vt0=-2.0 beta=1.0e-4
```

Loads the *mos.va* Verilog-A file.

```
.hdl mos_ideal.va module=jfet alias=ideal_jfet
.hdl mos_simple.va module=jfet alias=simple_jfet

x1 d1 g1 s1 ideal_jfet vt0=-2.0 beta=1.0e-4 lambda=0.01
x1 d2 g2 s2 simple_jfet vt0=-2.0 beta=1.0e-4 lambda=0.01
```

Shows how the same module name in two different source files can be loaded using the `alias_name` parameter.

Related options

"MACMOD=[1|2|3|0]" on page 954, "SPMODEL=subckt_name" on page 1020, "VAMODEL=mod_name" on page 1020, and "VAOPTS=options_string" on page 1021

Related command-line flags

"-hdlpath veriloga_path" on page 48,
"-spmodel <subckt_name1> -spmodel <subckt_name2> ..." on page 54, and
"-vamodel <mod_name1> -vamodel <mod_name2> ..." on page 56

-compat flag

In `-compat` mode, the syntax is different:

```
.HDL "filename" [module_name] [module_alias]
```

.HIER

Changing the Hierarchy Separator

```
.HIER .|/|<char>
```

By default the hierarchical separator in Eldo is the '.' character. The Verimix interface imposes the use of the '/' character for the hierarchical separator. Therefore, for Verimix integration, '/' is allowed as the hierarchical separator.

Please refer to “.A2D” on page 528 and “.D2A” on page 577.

The rule defined in “Hierarchical Management” on page 681 applies, however, the hierarchical separator differs.

There is no impact on the Eldo naming convention. In the Eldo description part, all the hierarchical names use the default hierarchical separator (by default '.'). To change the hierarchical separator in Eldo, use the **.HIER** command, for example the line:

```
.HIER /
```

changes the default hierarchical separator '.' to '/'.

- char

  It is possible to specify any character as the hierarchical separator. However, care must be taken as there may be possible side effects. One restriction is that this character should not appear elsewhere in instance or net names.

Example for the **.A2D** command with Verimix:

```
in the eldo description part | in the verilog description part
-----------------------------+-------------------------------
                             |
XIC1 OUT1 OUT5 My_Subckt     |
...                          |
                             |
.SUBCKT My_Subckt IN OUT     |
...                          | ...
 .A2D digital_name eldo_name |  $vmx_define_export(test.top.ic1.out2,
 +MOD=model_a2d              |                 "XIC1/DIGITAL_NAME");
...                          | ...
.ENDS My_Subckt              |
```

The hierarchical name of the eldo node `eldo_name` is `XIC1.DIGITAL_NAME`. In this case the name sent from Eldo to the Verimix interface is `XIC1/DIGITAL_NAME`.

Example for the **.D2A** command in Verimix:

```
in the eldo description part | in the verilog description part
```

```
------------------------------+------------------------------
                              |
XIC1 OUT1 OUT5 My_Subckt      |
...                           |
                              |
.SUBCKT My_Subckt IN OUT      |
...                           | ...
 .D2A digital_name            |  $vmx_define_import(test.top.ic1.out4,
 +VAnalogSource               |                 "XIC1/DIGITAL_NAME");
 +MOD=model_d2a               | ...
...                           |
.ENDS My_Subckt               |
                              |
```

In this case the name sent from Eldo to the Verimix interface is `XIC1/DIGITAL_NAME`.

Note

If the subcircuit `My_Subckt` is instantiated in another subcircuit, the hierarchical name will look like `<...>/XIC1/DIGITAL_NAME`.

Hierarchical Management

The hierarchical management for the `digital_name`, which is implicitly generated from the `eldo_name` by Eldo is allowed. To allow hierarchical netlist generation, the implicit `digital_name` generated by Eldo for the synchronizer takes into account the hierarchical context, for example:

```
XIC1 OUT1 OUT5 My_Subckt
...
.SUBCKT My_Subckt IN OUT
...
 .A2D eldo_name
 +MOD=model_a2d
...
.ENDS My_Subckt
```

The hierarchical name of the Eldo node `eldo_name` is `XIC1.ELDO_NAME`. However, the implicit `digital_name` sent to the synchronizer is `XIC1.ELDO_NAME` and *not* `eldo_name`.

.IC

Initial Transient Analysis Conditions

```
.IC V(NN)=VAL [SUBCKT=subckt_name] {V(NN)=VAL [SUBCKT=subckt_name]}
```

Used to fix node voltages for the duration of a DC analysis. If the `UIC` parameter is also present (in the **`.TRAN`** command) no DC analysis is performed and the voltages are initialized as defined in the **`.IC`** command. All other voltages on nodes not initialized in the **`.IC`** command are determined by Eldo itself. During subsequent analysis (transient), the node voltages are freed of their initial values, and may therefore assume different values.

The **`.IC`** command on devices is interpreted only on C and L devices. For all other devices, the **`.IC`** command is ignored.

By default, Eldo behaves like Spice 2g6 regarding **`.IC`** commands and **`IC`** parameters specified on devices: **`.IC`** commands are taken into account only for DC done before Transient analysis, or when **`.TRAN ... UIC`** is specified. However, **`IC`** parameter specifications on devices are taken into account only in the case of **`.TRAN ... UIC`**. Specify option **`ICDC`** for **`.IC`** commands (and **`IC`** parameter specifications on devices when option **`ICDEV`** specified) to be taken into account for any DC analysis. Specify option **`ICDEV`** for **`IC`** parameters specified on devices and **`.IC`** commands to be handled the same way.

Note

By default, the first IC specification has precedence over subsequent IC specifications. Setting option **`LICN`**, the last IC specification will have precedence.

Parameters

- **`V(`**`NN`**`)`**

  Voltage at the node `NN` in volts.

- **`SUBCKT`**`=subckt_name`

  If specified it will fix the voltage of the preceding node in all instances of the subcircuit `subckt_name`.

Examples

```
.dc
.ic v(2)=3v v(4)=-2v
```

Specifies that for the duration of the DC analysis, the voltages at the nodes `2` and `4` be fixed to 3 and –2V respectively. Other circuit node voltage values are computed by Eldo.

```
.tran 1ns 100ns uic
.ic v(n3)=7v v(n4)=2v v(n5)=-3v v(x1.222)=0v
```

Specifies that no DC analysis should be performed and that at the beginning of the transient analysis the voltages at the nodes `n3`, `n4` and `n5` be set to 7V, 2V and –3V respectively, while node `222` of subcircuit `x1` is set to 0V. Other node voltages are calculated by Eldo.

```
.dc
.ic v(2)=3v SUBCKT=sub1 v(4)=-2v SUBCKT=sub2
```

Specifies that for the duration of the DC analysis, the voltages at node `2` of subcircuit `sub1` and node `4` of subcircuit `sub2` will be fixed to 3 and –2V respectively. Other circuit node voltage values are computed by Eldo.

Related options

"ICDC and ICDEV" on page 978; "LICN" on page 979; "NOLICN" on page 981.

.IGNORE_DSPF_ON_NODE

Ignore DSPF on Specified Node

```
.IGNORE_DSPF_ON_NODE {NODE}
```

Used to force Eldo to ignore parasitic elements on a specified node when a DSPF file has been included using the **.DSPF_INCLUDE** command. All parasitics attached to the specified node will be removed: those inside the specified node (net) and those detailed inside a different node but connected to the specified node.

This command is ignored when **DEV=DSPF** is specified on the **.DSPF_INCLUDE** command.

Parameters

- NODE

  Specifies node(s) for which the parasitics given in the DSPF file will be ignored.

Example

```
.DSPF_INCLUDE testspf.spf
.IGNORE_DSPF_ON_NODE n1 n2
```

For nodes n1 and n2 parasitics as specified in DSPF file *testspf.spf* will be ignored.

In the example DSPF file snippet below, if **.IGNORE_DSPF_ON_NODE** N1 is specified Eldo ignores the coupling capacitance CC1. This is because it connects to the N1 net even though it is detailed inside the N2 net.

```
*|NET N1
*|S N1:1
C1 N1:1 0 0.02p

*|NET N2
*|S N2:1
CC1 N2:1 N1:1 0.01p
```

.INCLUDE

Include a File in an Input Netlist

```
.INC[LUDE] FNAME
```

Inserts the contents of a file into a circuit description file. Including a file is the same as typing the included file's text directly into the circuit description file. Included files may not contain title lines (comments may be used). A **.END** statement in an included file marks only the end of the included file.

Eldo searches for the specified file in a specific order. This works in the same way as the **.LIB** command:

1. Absolute path
2. Parent directory
   Directories of the files in the calling hierarchy (files which contain the **.INCLUDE**/**.LIB** statement), the directory of the last calling file being searched first
3. Current directory
   Directory from which Eldo was started
4. Search path
   Specified using the `-searchpath` flag or **.OPTION SEARCH**=`path` within the netlist. If both are specified, then the contents of `-searchpath` is searched first.

For more information see "Search path priorities" on page 694.

Nesting of **.INCLUDE** commands is allowed. **.INCLUDE** files are included only once even if the **.INCLUDE** statement appears several times. However, when the **.INCLUDE** is located inside a SUBCKT, the inclusion will occur.

The **.INCLUDE** command can also be used to include DSPF files that contain the complete netlist (active elements and parasitics). Subcircuits can be instantiated using either the **.TOPCELL** command or an `X` instance.

Use option **CONTINUE_INCLUDE** to specify that continuation lines with + as the first character apply to the **.INCLUDE** command in the file.

To replace one file by another one when using the Eldo re-run facility, use the option **ALTINC**. This forces Eldo to replace the first **.INCLUDE** statement found in an input netlist by the first **.INCLUDE** statement found in the **.ALTER** section of the netlist.

By default the content of **.INCLUDE** commands specified in **.ALTER** blocks is treated by Eldo as if the content is written at full length in the netlist and substitutes accordingly. In previous versions (pre-v6.9), the **.INCLUDE** commands specified in **.ALTER** blocks were added (extended) to the nominal circuit. For backward compatibility, specify option **NOALTINCEX** to switch back to this mechanism.

Related Options

ALTINC, see "ALTINC" on page 951, **CONTINUE_INCLUDE**, see "CONTINUE_INCLUDE" on page 952, **NOALTINCEX**, see "NOALTINCEX" on page 955,.

Parameters

- FNAME

  Name of the file to be included. This name may be any character string which is a legal path name. The filename can be written in either upper or lower case letters. It is possible to have a mixture of both.

Examples

```
.INCLUDE circuit_add_on.cir
```

Specifies that the contents of the *circuit_add_on.cir* file be included in the circuit description file. In this case, the included file is resident in the same directory as the circuit description file.

```
.INCLUDE /users1/examples/circuit2.cir
```

Specifies that the */users1/examples/circuit2.cir* file is to be included in the circuit description file.

```
TITLE
.INCLUDE foo
.INCLUDE foo  !this include will be ignored.
...
.END
```

In the example above, the second **.INCLUDE** statement is ignored.

```
TITLE
.SUBCKT S1..
   .INCLUDE foo
.ends
.SUBCKT S2
   .INCLUDE foo
.ends
.end
```

In the example above, the file named foo will be included twice. The **.INCLUDE** statements have been located inside subcircuits.

The following example shows how to specify that continuation lines with + as the first character apply to the **.INCLUDE** command in the file. If the main netlist has the two lines:

```
.include file.inc
+ b=2
```

With the **CONTINUE_INCLUDE** option specified, if file *file.inc* contains as its last line:

```
.param a=1
```

Eldo would interpret this as:

```
.param a=1
+ b=2
```

The following example shows how to replace one included file by another one when using the Eldo re-run facility, with the option **ALTINC**. The *new_models* file will be included in the re-run simulation instead of *models*.

```
.include models
.option altinc
r1 1 0 rval
v1 1 0 dc 1
.extract dc i(r1)
.dc
.alter
.include new_models
```

By default the content of **.INCLUDE** commands specified in **.ALTER** blocks is treated by Eldo as if the content is written at full length in the netlist and substitutes accordingly. In previous versions (pre-v6.9), the **.INCLUDE** commands specified in **.ALTER** blocks were added (extended) to the nominal circuit, for example:

```
i1 1 0 1
r1 1 0 1
.op
.extract dc v(1)
.alter
.include altincex.a1
.end
```

assuming file *altincex.a1* contains the line r1 1 0 2, for the first ALTER run Eldo will substitute r1 1 0 1 by r1 1 0 2 in the netlist (behaves as if line r1 1 0 2 was in the netlist). For Eldo versions before v6.9, Eldo assumed two resistors in parallel to the I1 device. For backward compatibility, specify option **NOALTINCEX** to switch back to the old mechanism.

.INIT

Initial Digital Circuit Conditions

```
.INIT NODE [DC=VAL] TI TS VALI {TI TS VALI}
```

This command is used when initializing digital circuits. Between the times `TI` and `TS`, Eldo forces the node `NODE` to the voltage level specified by `VALI`. If a DC value is also specified, the simulation is started with the specified value assigned to the node. Outside the specified time limits the node is computed as a normal node.

Parameters

- `NODE`

  The name of the node to be initialized with a voltage level.

- `TI`

  The time from which the node is to be initialized to the specified voltage level.

- `TS`

  The time beyond which the voltage initialization on the node should be stopped.

- `VALI`

  The voltage level to which the node should be initialized, in volts.

- **`DC`**`=VAL`

  Voltage level to which the node should be initialized at the start of the simulation period. Optional.

Example

```
.init n1 dc=2 5u 10u 5
```

Specifies that the node `n1` be initialized to 5 V between the time period 5 and 10 µs. The node is initialized to 2 V at the start of the simulation.

.IPROBE

Current Probe

```
.IPROBE LABEL=vname [DIRECTION=POS|NEG ] pinref1 {pinrefn}
```

Use this command to probe current between pins. This is equivalent to inserting a zero voltage source between a node and a specified pin of a device. However it does not require netlist modification, which means for users working in a UI schematic environment, the netlister does not have to be run again.

Inserting a zero voltage source can be useful to measure current entering objects, or for some extra commands such as **.LSTB**.

You can access the probed current with the expression IPROBE(vname).

IPROBE reports the SUM of the currents through the referenced pins. The "positive flow direction" is from the connection node into the device pin. Pins are referenced via the device pin INDEX number, for example r1.1 and r1.2 are the first pin of resistor r1 and the then second pin of resistor r2 respectively. All referenced pins must connect to the same (possibly hierarchical) node. If there are N pins connected to a node, at most N-1 pins can be specified in the pin list.

Parameters

- **LABEL**=vname

  Name of the voltage source to add. Current through this voltage source can be read using `IPROBE(vname)`.

- **DIRECTION**=**POS**|**NEG**

  This is **POS** by default. The voltage source is added between the original nodes referred to by the `pinref` list and a new node which will be created by Eldo, the name of which will be constructed from the prefix `IPRB#` followed by `vname`. If direction is **NEG**, the voltage source is reversed.

- pinref

  List of pin references connected to the same node. Syntax is `device_name.pin_index`. The number of pin references is not limited. All pin references must refer to the same original netlist, otherwise an error will be displayed. The pin references cannot make reference to ADMS devices, because such ADMS objects (Y instances) can be hierarchical devices, and this situation is not handled.

Notes

.IPROBE is used to insert a current probe (zero voltage source) between a node and a selected number of pin objects. This pin list does *not* correspond to the pin list of the new zero voltage source. Imagine you have:

```
R1 A B 1k
R2 B C 1K
```

If you want to insert a current probe between R1 and R2, the syntax will be:

```
.IPROBE label = I R1.2
```

The following statement is incorrect:

```
.IPROBE label = I R1.2 R2.1
```

In both cases, Eldo will first create a new node named, for example, NEW. Eldo will then create a zero voltage source named, for example, VI. Eldo will connect node B to the first pin of VI, the second pin of I will be connected to NEW, and Eldo will then connect the pin list specified to the node NEW. With the first syntax above, only pin 2 of device R1 will be reconnected to NEW, but for the second syntax, both the second pin of R2 and the first pin of R2 will be reconnected to node NEW. In that second case, the original net B will therefore be connected only to pin NEW, which is not what was required.

The equivalent netlist would be in case 1:

```
R1 A B 1k
VI B NEW 0
R2 NEW C 1K
```

The equivalent netlist would be in case 2:

```
R1 A NEW 1k
VI B NEW 0
R2 NEW C 1K
```

and B here is a dangling node.

Example

```
V1 1 0 1
r1 1 2 1
r2 3 1 1
r3 2 0 1
r4 3 0 1
r5 1 0 1

.dc
.IPROBE label=ux r1.1 r2.2
.print dc iprobe(ux)
.extract dc iprobe(ux)
```

The actual pin indicated by the pin reference list here is the node named “1”. It is the first pin of device R1 or the second pin of device R2. Eldo will insert a current probe (zero voltage source) between node 1 and a new node named `IPRB#UX`. The first pin of R1 and the second pin of R2 will be disconnected from node 1 and reconnected to the new node `IPRB#UX`. The IPROBE(UX) statement returns the sum of the current inside R1 and R2.

```
v1 1 0 1
r1 1 2 1
x1 2 3 foo
r3 3 0 1
.subckt foo 1 2
r1 1 2 1
.ends
.iprobe label=ip x1.1
.print tran iprobe(ip)
.tran 1n 5n
.end
```

This example shows how to insert a current probe with a subcircuit. Eldo will insert a zero voltage source between node 1 and a new node named `IPRB#IP`. The first pin of subcircuit x1 will be disconnected from node 1 and reconnected to the new node `IPRB#IP`. The IPROBE(IP) statement returns the sum of the current inside x1.

```
v1 1 0 sin(0 1 1e9)
r1 1 2 1k
r2 2 3 1k
r3 2 0 1k
.tran 1p 1n
.iprobe label=test r1.2 r3.1
```

r1, r2, and r3 are connected to node 2. The iprobe command reports the total current through r1 and r3 with current “direction” from node 2 into the pins.

.LIB

Insert Circuit Information from a Library File

```
.LIB [KEY=KNAME] FNAME [LIBTYPE]
.LIB LIBTYPE
```

This command is used to insert model or subcircuit definitions into an input netlist from a library file. Eldo includes the whole contents of the file specified. Although many subcircuits, models and parameters may be defined but not used, it is quicker than picking out the individual instances required.

If there are several definitions of the same instance defined in a block, Eldo will stop and produce an error message.

Nesting of **.LIB** commands is allowed. For more information on nested **.LIB** commands see "Library nesting" on page 695.

Eldo searches for the specified library file in a specific order. First it checks the directory the netlist is in, followed by the directory Eldo was started in, and then in the library file parent directory. For more information, see "Search path priorities" on page 694.

If a library file contains corners, the LIBTYPE parameter is required in the netlist (see second syntax definition), otherwise Eldo will exit with an error. If a library file has been referenced without a corner, Eldo does not expect the syntax **.LIB** LIBTYPE inside the library file, and interprets all the **.LIB** commands as **.LIB** FILENAME. Avoid this by referencing any library file containing corners with the LIBTYPE parameter.

Libraries can be accessed directly from model and subcircuit description lines. This allows the properties of the model, or subcircuit, to be called from another file or directory. The syntax is as follows:

```
.MODEL LIB FNAME MNAME [LIBTYPE]
.SUBCKT LIB FNAME SNAME [LIBTYPE]
```

where MNAME and SNAME are the model or subcircuit names respectively.

It is also possible to use the **.LIB** command in interactive mode in SimPilot. See "Interactive mode" on page 695 for details.

If a parameter P is referred to in a netlist but not defined, Eldo searches for P in the **.LIB** files. Only global **.PARAM** definitions are considered. Parameter declarations within a **.SUBCKT** definition will not be considered outside that subcircuit.

Parameters

- FNAME

  Name of the library file to be searched. This name may be any character string which is a legal path name. FNAME must not end with *<circuit_name>.lib* or *<circuit_name>_eldo.lib* since Eldo uses this type of filename for temporary files which are later deleted automatically by the Eldo script, that is, in the circuit file *mycircuit.cir*, the file *mycircuit_eldo.lib* must not be used.

Note

A filename specified in the **.LIB** statement may be enclosed in single quotes to preserve compatibility with other simulators. When a character string is enclosed in single quotes, the case of the string remains unchanged, that is, no conversion to upper or lower case, allowing a mixture of the two.

- **KEY**=KNAME

  Used in conjunction with the **.ALTER** or **.STEP** commands. KNAME is the key name used to identify the library files and variants (used in subsequent simulations) that may be replaced with the **.ALTER** command. For more information on the key parameter see the "Key parameter example" on page 696.

- LIBTYPE

  Name of a library variant to be used. If a library file contains corners, this parameter is required for the **.LIB** statement in the netlist. For more information on library variants see "Library variant management" on page 694.

Library management mechanism

A number of different mechanisms are available to manage libraries.

The library management changed in v5.8 compared to previous versions, in the way that the entire content of the selected library is included in the database. In pre-v5.8 versions, only the missing **.SUBCKT**, **.PARAM** or **.MODEL** definitions were extracted from the library.

In v5.9 and upwards, the change for v5.8 remains, but the **.LIB** command is acted upon immediately, that is, the content of the library is incorporated as soon as the command is parsed. In v5.8, the contents of the different libraries were incorporated only upon completion of reading the input file. In pre-v5.8 versions, libraries were re-read repeatedly to load missing definitions.

Note

The pre-v5.8 version of library management is no longer supported.

An option, **INCLIB**, is available to use the alternate v5.8 mechanism.

v5.8 mechanism

To use the Eldo v5.8 mechanism of library management, specify the option **INCLIB**.

With libraries containing thousands of definitions which are all likely to be used in a design, the procedure used in pre-v5.8 versions of Eldo, was far too slow. Specifying the v5.8 mechanism means Eldo will include the whole content of the **.LIB** (or the full content of what is inside **.LIB .ENDL** blocks when variants are specified on the **.LIB** card). This procedure is faster, even if some models, subcircuits or parameters are defined but not used.

However, there are four side effects to this mechanism:

- When there are several definitions defined in the block that Eldo will include, the simulation will stop on error (previous versions accepted this but used only the first definition—it was however not safe to potentially have two definitions for the same entity).
- Eldo includes the full content of the **.LIB** card, which means Eldo interprets any commands/options which are specified in **.LIB**. With the pre-v5.8 implementation, these commands were not acted upon because they were not looked for.
- Parameters and models are defined in the order which corresponds to their order in the library. With the pre-v5.8 implementation, the order was the order of usage (first one used was the first one defined). Therefore, the Monte Carlo series associated to each entity will not be the same in both cases.
- If the filename specified in the **.LIB** does not exist, Eldo exits with an error.

In all cases, the content of what is actually included by Eldo appears in the ASCII output file.

Search path priorities

The order of directories in which include and library files are searched for is:

1. Absolute path
2. Parent directory
   Directories of the files in the calling hierarchy (files which contain the **.INCLUDE**/**.LIB** statement), the directory of the last calling file being searched first
3. Current directory
   Directory from which Eldo was started
4. Search path
   Specified using the `-searchpath` flag or **.OPTION SEARCH**=`path` within the netlist. If both are specified, then the contents of `-searchpath` is searched first.

For more information on `-searchpath`, see “-searchpath path_list” on page 53 or “SEARCH=path1[{:path2}]” on page 1022. For an example, see the “Search path priorities example” on page 696.

Library variant management

Library variants such as `best`, `worst`, and `typical` process variation, may be handled using the following command:

```
.LIB FNAME LIBTYPE
```

Within the library `FNAME`, you must then have sections defined by:

```
.LIB <libtype>
...
.ENDL
```

For an example see the “Library variant management example” on page 697.

Interactive mode

It is possible to use the **`.LIB`** mechanism in SimPilot. The netlist must contain one or several lines of type:

```
.LIB KEY=kname FNAME [LIBTYPE]
```

In interactive mode, the following commands can be issued:

```
.LIB key=toto mymod.tech nominal
.LIB key=toto mymod.tech worst
.LIB key=toto mymod2.tech
```

Eldo will search for the `kname` to identify which library must be replaced. Limitations of the **`.LIB`** command when issued in interactive mode are as follows:

- Subcircuits are not replaced.
  When **`.LIB`** is specified in the input file, it can be used to select a subcircuit to be included in the design. Since taking a subcircuit from another library than the library specified in the input file could result in a change of topology (and changing topology of the current design is strictly impossible), Eldo would issue an error whenever the user attempts to substitute one subcircuit for another.
- Switching between GUDM and non-GUDM models is prohibited. Similarly, switching between GUDM models is prohibited.
- Only MOS, BJT, DIODE, JFET, R, L and C **`.model`** commands can be substituted. Attempting to substitute other kinds of models will lead to an error.

Library nesting

Eldo libraries may themselves contain other libraries. For example, consider a library *lib.lib*:

```
.lib best
.lib mos.lib best
.lib bip.lib best
.endl best
.lib typ
.lib mos.lib typ
.lib bip.lib typ
.endl typ
```

If a netlist contains the command:

```
.LIB lib.lib best
```

then Eldo will browse the *mos.lib* and *bip.lib* which are surrounded by **.lib** best and **.endl** best.

Examples

The following example specifies that any model and subcircuit information missing in the input netlist should be taken from the file *circuit1.cir*.

```
.LIB circuit1.cir
```

The following example specifies that any model and subcircuit information missing in the input netlist should be used from the file *circuit2.cir* in the directory `/users1/examples`.

```
.LIB /users1/examples/circuit2.cir
```

Search path priorities example

The following example is related to search path priorities and shows the order in which directories are searched. If you have the following command in a netlist file:

```
.LIB ./foo/myfile.lib
```

and if in the library file `myfile.lib`, you have:

```
.LIB toto
```

then if file `toto` is not found in the netlist directory or parent directory, it will be searched for in the current directory from which Eldo was started.

In the case of a nested **.LIB**/**.INCLUDE**, it would search the directory of the library file containing the **.LIB**/**.INCLUDE** statement.

Key parameter example

The example below demonstrates the use of the **KEY** parameter in conjunction with the **.ALTER** command.

```
...
.lib key=K1 /work/bip/mymod typ
.lib key=K2 /work/mos/mymod typ
...
.alter
.lib key=K2 /work/mos/mymod best
.alter
.lib key=K2 /work/private/mymod typ
.end
```

This command sequence causes three simulations to be performed always with library `/work/bip/mymod typ`. In the first simulation, library `/work/mos/mymod typ` is used. In the second simulation, library `/work/mos/mymod best` is used. In the third simulation, library `/work/private/mymod typ` is used.

Library variant management example

The next example relates to Library variant management.

Library *mos.lib*:

```
.lib best
.model MN nmos level=3 vt0=0.5
.endl best
.lib typ
.model MN nmos level=3 vt0=0.75
.endl typ
.lib worst
.model MN nmos level=3 vt0=1.0
.endl worst
```

Circuit Netlist:

```
lib case management
.lib mos.lib typ
m1 vdd g 0 0 MN l=1.2U w=5U
vdd vdd 0 5
vg  g   0 0.8
.op
.end
```

.LOAD

Use Previously Simulated Results

```
.LOAD [FILE=]filename
```

This command takes a set of voltages previously saved using the **.SAVE** command, and inserts these as **.NODESET**, **.GUESS**, or **.IC** commands depending on how the LOAD file was created. It is also possible to have multiple occurrences of the **.LOAD** command in an input netlist.

This command works in a similar way to the **.USE** command. Simply, the attribute **NODESET**/**IC**/ **GUESS** need not be specified in the command since that information was dumped when creating the file with the **.SAVE** command.

See also ".USE" on page 926.

Parameters

- `[FILE=]filename`

  Filename into which the DC values were saved via the `.SAVE file_name DC` command.

.LOOP

Insert a Feedback Loop

```
.LOOP INPUT OUTPUT [R|C|I|V VALUE] [DISCONNECT=DEV_NAME] [KEEPINPUT]
```

Inserts a feedback loop between the input and output nodes of an op-amp during transient analysis. This is, therefore, useful in obtaining both the Open and Closed loop characteristics of a circuit in the same simulation run. The following devices can be inserted in the feedback loop:

- Resistor.
- Capacitor.
- Independent Voltage Source.
- Independent Current Source.

If no device is specified, Eldo will insert a zero-voltage source between `INPUT` and `OUTPUT`. A second operation which is performed in the Closed-loop mode is to disconnect the voltage source (**`V`** or **`E`** element) which is applied between node `INPUT` and **`GND`**, if present. This feature can be disabled using the keyword **`KEEPINPUT`** in the **`.LOOP`** command.

In addition to the above operation, a device which is active in Open-loop mode may be disconnected during simulation in Closed-loop mode using the **`DISCONNECT`** keyword.

Parameters

- `INPUT`

  Input node of the operational amplifier which is to be connected in closed loop.
- `OUTPUT`

  Output node of the operational amplifier which is to be connected in closed loop.
- **`R`**

  Keyword indicating a feedback loop with a resistive load.
- **`C`**

  Keyword indicating a feedback loop with a capacitive load.
- **`I`**

  Keyword indicating a feedback loop with an independent current source.
- **`V`**

  Keyword indicating a feedback loop with an independent voltage source.
- `VALUE`

  Value of the feedback device.

- **DISCONNECT**

  Keyword indicating that a device in the feedback loop is to be disconnected.

- DEV_NAME

  Instance name of the device in the feedback loop which is to be disconnected.

- **KEEPINPUT**

  If specified, the voltage source (**V** or **E** element) which is applied between node INPUT and **GND** will not be disconnected, if present.

Note

The AC analysis conditions defined in a netlist are always simulated prior to any transient analysis conditions in that same netlist.

Example

```
r1 1 2 1k
c1 2 0 10p
c2 4 0 1p
r2 3 0 100
.model ampop opa level=2 voff=0 sl=50e06
+ cin=0 rs=10 vsat=5 gain=5000 fc=5000
v1 1 0 ac 1 pwl(4n 5 10n 0 20n 0 30n 5)
opa1 2 3 4 0 ampop
.loop 2 4 r 100
.ac dec 10 10e+4 10e+8
.tran 1m 100m
.plot ac vdb(4)
.plot tran v(4)
```

Specifies that during transient analysis of the circuit, a resistive load of 100Ω is inserted between the input node 2 and output node 4 of opa1. During AC analysis, no feedback loop is inserted between these nodes.

Note

The definition of two voltage sources v1 in the netlist. Eldo uses the AC voltage source definition for the AC circuit analysis and the transient voltage source definition for the transient circuit analysis.

.LOTGROUP

Share Distributions

```
.LOTGROUP group_name[/distrib_type]=val[%]
```

This syntax allows different entities to share the same distribution.

Anywhere Eldo accepts **LOT** or **DEV** specification, one can specify **LOTGROUP**=group_name.

Parameters

- group_name
  Name of group.
- distrib_type
  This can be one of the following options:

  UNIFORM
  Uses a uniform distribution (default).

  GAUSS
  Uses a Gaussian distribution.

  <dist_name>
  Selects a user defined distribution named dist_name.

  If no distribution type is selected, it will default to **UNIFORM**.

Examples

```
.LOTGROUP my_lot_group=14%
.LOTGROUP my_lot_group/uniform=14%
.LOTGROUP my_lot_group/gauss=12%
.LOTGROUP my_lot_group=(nor,-12%,10%)
.LOTGROUP my_lot_group/my_distrib_name=8%
```

This works together with:

```
.DISTRIB my_distrib_name (-1,0) (-1,0.5)
+                        (-0.4,0.5) (-0.4,0)
+                        (0.4 0) (0.4 0.5)
+                        (1 0.5) (1,0)
```

For more information see ".DISTRIB" on page 614.

All entities which refer to the same "lotgroup" will share the same distribution.

```
.LOTGROUP my_lot_group=14
.MCMOD MOD1 TOX LOTGROUP=my_lot_group
.PARAM P1=1 LOTGROUP=my_lot_group
```

In this example, the same random number, between +14 and -14, will be used for updating `P1` and **`TOX`** of model `MOD1`.

.LSTB

Loop Stability Analysis

```
.LSTB SOURCE_NAME
```

This command improves the analysis of circuit stability. The classical method for stability analysis is to break the feedback loop at an appropriate point on AC analysis, while maintaining correct DC conditions. This means that the loop must be terminated with the appropriate impedance it ‘sees’ looking at the loop input. Obtaining this impedance value is not always a simple task.

The **.LSTB** command measures the loop gain by successive injection (Middlebrook Technique). A zero voltage source is placed in series in the loop: the first pin of the voltage loop must be connected to the loop input, the other pin to the loop output. The name of this voltage source is given in the **.LSTB** card, and the loop gain can be displayed using keywords **LSTB_DB**, **LSTB_P**, **LSTB_R**, **LSTB_I**, **LSTB_M** (see meanings in table below) in any **.PLOT**/**.PRINT**/ **.PROBE**/**.EXTRACT** commands.

Table 10-15. LSTB Output Formats

Format	Meaning
DB	Magnitude in dB
M	Magnitude
P	Phase
R	Real part
I	Imaginary part

Example

aopstb.cir (provided in: *$MGC_AMS_HOME/examples/eldo/*)—this circuit is derived from *aopbou.cir* and *aopalt.cir* which are also provided.

- Split node S into node S and SL, and insert the VSTB source:

```
M7 B SL C VSS mod1 W=130U L=4U
VSTB SL S
```

- Invoke LSTB analysis:

```
.LSTB VSTB
```

- Output the LSTB analysis:

```
.plot ac lstb_db
.plot ac lstb_p
```

.MALIAS

Model and Subcircuit Name Mapping

```
.MALIAS actual_name alias_name
```

This command can be used to map a model (or subcircuit) name in a netlist to a model (or subcircuit) name specified in a **.MODEL** (or **.SUBCKT**) statement. This is similar to the .DEFMOD command, except the arguments are reversed.

The mapping works if the **.MALIAS** is placed before any use of the string `alias_name`.

Parameters

- `actual_name`

  Model name defined in **.MODEL** statement, or subcircuit name defined in **.SUBCKT** statement.

- `alias_name`

  Model (or subcircuit) name specified in a netlist.

Example

```
.model model1 r tc1=2 tc2=1
.malias model1 modalias
r1 1 0 modalias r=1k
```

Model name `model1` is aliased to `modalias`.

```
.subckt myres in out rr=1
r1 in out r=rr
.ends

.malias myres myres_alias
x1 in out myres_alias r=20k
```

Subcircuit name `myres` is aliased to `myres_alias`.

.MAP_DSPF_NODE_NAME

Map Eldo Node to DSPF Node

```
.MAP_DSPF_NODE_NAME LOGICAL=ELDONAME DSPF=NEWNAME
```

This command can be used if the DSPF file specifies for a node to be replaced causing references to nodes in commands such as **.PLOT**, **.PROBE** and **.EXTRACT** to become unusable. Eldo will map all of the node references to the new DSPF node name. For information on loading DSPF files see “.DSPF_INCLUDE” on page 617.

Parameters

- **LOGICAL**=ELDONAME

  Name of node to be mapped.

- **DSPF**=NEWNAME

  DSPF node name to replace all references to **ELDONAME**.

.MC

Monte Carlo Analysis

```
.MC RUNNO [OUTER] [OV] [SEED=integer_value] [NONOM] [ALL]
+ [VARY=LOT|DEV] [IRUN=val] [NBBINS=val] [ORDMCS] [MCLIMIT]
+ [PRINT_EXTRACT=NOMINAL|ALL|run_number] [SIGBIN=val]
+ [MAXABSBIN=val] [MAXRELBIN=val]
+ [MONITOR] [AUTOSTOP=expression]
+ [SAVE=mc_file] [RESTART=mc_file]
```

Please refer to the Monte Carlo Analysis chapter for further information.

The Monte Carlo system may be implemented for DC, AC and transient analysis and is useful to obtain statistical information derived from estimates of the random variability of all circuit components. Model parameters may be specified with nominal and tolerance values. The Monte Carlo analysis system carries out multiple simulation runs, each run using model and device values differing from the nominal one within the specified tolerance limit, the variation being a simulated random variable satisfying a specified distribution (uniform, Gaussian, or user-defined).

This kind of analysis is useful in yield prediction and synthesis. When Monte Carlo analysis has been requested, information is added to each item plotted or printed regarding minimum and maximum values.

.MPRUN can be used to take advantage of multi-processor machines for the **.MC** command. Please see ".MPRUN" on page 729 for further information.

The keyword, **STATISTICAL**= 0|1, can be specified on X instances, device declarations, or on **.SUBCKT** definitions, to specify whether any statistical variation due to Monte Carlo analysis can be applied to the specified entities. If **STATISTICAL** is 0, the selected devices will keep their nominal values. If **STATISTICAL** is 1, the selected devices have statistical variation applied. The global default can be specified via option **STATISTICAL**= 0|1. Default is 1.

The standard deviation is calculated for the output specified in the **.MC** command for each Monte Carlo run. At the end of the Monte Carlo analysis, a print-out is made of the worst case value. Furthermore, all the Worst Case model parameters, together with the values of the dipoles are printed.

Two standard deviation results are provided: "standard deviation" is the RMS value of the deviation of output with respect to the average value; "standard deviation based on nominal run" is the RMS value of the deviation of output with respect to the nominal run.

$$(\text{standard deviation based on nominal run})^2 = (\text{standard deviation})^2 - (\text{nominal value})^2 + (\text{average value})^2$$

Note

Multiple sensitivity and statistical analyses cannot be used simultaneously. Specifically, **.MC**, **.DCMISMATCH**, and **.WCASE** are exclusive. Only one of them can be specified in a netlist. Additionally, the **.MC** command may not be used in a netlist together with transient noise analysis (**.NOISETRAN**).

The results of Monte Carlo analysis runs are output using the **.EXTRACT**, **.PRINT** and **.PLOT** commands.

By default, **.MC** used without the **ALL** keyword disables option **PROBEOP**, option **PROBEOP2**, and PSF OP information. Use option **MC_NOMINAL_OP** to save OP results for the nominal run.

Parameters

- RUNNO

  Number of simulation runs.

- **OUTER**

  When there are both **.STEP** and **.MC** commands, Eldo performs a full Monte Carlo analysis for each point of the **.STEP** command. If the keyword **OUTER** is specified on the **.MC** command the nesting of the simulations will be inverted. Instead, a **.STEP** will be performed at each Monte Carlo run (the **OUTER** keyword must be placed after the specification of the number of Monte Carlo runs). **.MC OUTER** does not work with a variation in temperature with the **.TEMP** command (for example **.TEMP** 0 39 80) or **.STEP TEMP** command, but does work with a single temperature definition with **.TEMP** (for example **.TEMP** 70).

- OV

  Requests the output of a node voltage or current through a voltage source. These output values are used as reference for the Worst Case analysis.

  The syntax for the voltage or current output is as follows:

  I(Vxx[, **V**yy]**)**

  Specifies the current difference between the voltage sources **V**xx and **V**yy. If **V**yy and the comma are omitted, the current through **V**xx will be printed.

  V(N1[, N2]**)**

  Specifies the voltage difference between nodes N1 and N2. If N2 and the preceding comma is omitted, ground is assumed.

  The following AC analysis output commands are also available:

```
VDB(N1[, N2])      IDB(Vxx[, Vyy]) IGD(v_source)
VI(N1[, N2])       II(Vxx[, Vyy])  VGD(node_name)
VM(N1[, N2])       IM(Vxx[, Vyy])
VP(N1[, N2])       IP(Vxx[, Vyy])
VR(N1[, N2])       IR(Vxx[, Vyy])
```

For more details on the above AC analysis output formats, refer to the AC Analysis printing and plotting specifications.

For each run, Eldo computes the standard deviation on the `OV` quantity and outputs it in the ASCII output file. The standard deviation (sigma) is computed in the following manner:

For transient simulation:
sigma = SUM(delta*delta*h) / TMAX
where `delta` is the difference between `OV` for nominal run and current run at current step, `h` is the time step for current step, and `TMAX` is the transient simulation duration.

For other analyses:
sigma = SUM(delta*delta) / nbpt
where `nbpt` is the total number of points in the simulation.

At the end of the MC simulation, Eldo indicates the run that generated the worst standard deviation as the worst case conditions in the ASCII output file. Here is an example of the output in the *.chi* file:

```
Standard Deviation for run   1: 1.621303E+00
Standard Deviation for run   2: 2.016042E+00
Standard Deviation for run   3: 8.805810E-02
...
Worst Case Conditions: Run Number  30
```

- **SEED**=integer_value

  This number is used to initialize the pseudo-random sequence of numbers. Running the **.MC** analysis twice with the same seed specified will provide the same simulation results.

- **NONOM**

  Nominal run for Monte Carlo analysis is bypassed. This option is used by Accusim. When **NONOM** is active, the **ALL** option of **.MC** is enabled as well.

- **ALL**

  If this optional parameter is declared, the waveform results of every Monte Carlo simulation run are stored in the output files (one set per `RUNNO`) in contrast to the usual nominal, maximum and minimum results.

- **VARY**=**LOT**|**DEV**

  Specifies that only a **LOT** or **DEV** specification is taken into account. When specifying **DEV** then **DEV** and **DEVX** variation is taken into account. Only one **VARY** specification can be set. By default, Eldo applies **LOT**, **DEV** and **DEVX** variation.

- **IRUN**=VAL

  Can be specified if a single, specific run, of a Monte Carlo analysis is required. `VAL` is an integer. `RUNNO` is still required to be specified, even though it is not used in that mode. Example:

  ```
  .MC 10 IRUN=3
  ```

Eldo will run a single analysis which corresponds to the third Monte Carlo analysis of the ten run series. If **IRUN** ≤ 0 it will be ignored and a warning generated.

- **NBBINS**=VAL

  Specifies the number of bins for the histogram produced when Monte Carlo analysis is used with **.EXTRACT** statements. Default is 10.

- **ORDMCS**

  Determines whether multiple MC parameters in the simulation share the same pseudo-random probability values or not. See ORDMCS in the Monte Carlo Analysis chapter for usage examples.

- **MCLIMIT**

  Specifies that all parameters with statistical distribution (**DEV**, **DEVX**, or **LOT**) will have their distribution modified to one of two deviation values. These values correspond to the maximum deviation of the original distribution as defined by the option **SIGTAIL**. For example, a parameter with a nominal value of 1.0, a statistical deviation of DEV/GAUSS=5%, and with option **SIGTAIL** at its default value of 4, the two values will be calculated as follows:

  ```
  1-(4*0.05)=0.8,    1+(4*0.05)=1.2
  ```

  This functionality can be useful to force Monte Carlo runs to use maximum deviation combinations. **MCLIMIT** affects all statistical parameters whatever their original distribution.

- **PRINT_EXTRACT**=**NOMINAL**|**ALL**|run_number

  Specifies for which run extracted values should be printed and written to output files.

  NOMINAL

  Only the nominal extracted value is printed (default).

  ALL

  Extracted values are printed for all runs.

  run_number

  Extracted values are printed only for the specified run_number (0 is the nominal run).

- **SIGBIN**=VAL

  Can be specified to truncate the histogram to a certain number of sigmas. Eldo will gather all the samples above "mean+SIGBIN×sigma" in the Above bin, and all the samples below "mean−SIGBIN ×sigma" in the Below bin. This might be useful when there are a few untypical samples that would otherwise corrupt the min and max of the histogram.

- **MAXABSBIN**=VAL

  Can be specified to simplify the histograms printed out in the .chi file, if they are difficult to read if most of the samples are in the same bin. If there are more than MAXABSBIN terms in a bin, then that bin will be expanded recursively, and a new histogram will be printed out for that bin. This corresponds to a zoom in each of the bins which contain too much samples. Default value is -1, that is, this option is not active by default.

- **MAXRELBIN**=VAL

  Can be specified to simplify the histograms printed out in the .chi file, if they are difficult to read if most of the samples are in the same bin. If there are more than MAXRELBIN% of the samples in the same bin, then that bin will be expanded recursively, and a new histogram will be printed out for that bin. This corresponds to a zoom in each of the bins which contain too much samples. Default value is -1, that is, this option is not active by default.

- **MONITOR**

  Monitor the evolution of certain quantities. Eldo will flush out from the *.wdb* file the average and standard deviation of extracts (**.EXTRACT**) and measurements (**.MEAS**) versus the Monte Carlo run index in order to see how these entities evolve. Eldo will also flush out of the *.wdb* file the expressions used in the **AUTOSTOP** criteria.

- **AUTOSTOP**=expression

  Automatically stops the Monte Carlo process based on the convergence of some or all of the quantities defined in the expression. The expression in the **AUTOSTOP** clause is a boolean expression using the **MCCONV** extracts as described in "Monte Carlo Convergence" on page 1218.

- **SAVE**=mc_file

  Saves any relevant information to a specific file. This files can then be used start a new session inheriting the results from the saved session. This save feature is disabled when multiple run commands are specified (**.STEP**, multiple **.TEMP**, **.AGE**, and **.MPRUN**).

- **RESTART**=mc_file

  Restarts a Monte Carlo simulation run from a previous session on the same design saved with the **.MC ... SAVE**=mc_file specification. If the file, mc_file, does not exist Eldo generates a warning and performs a normal Monte Carlo run. If a restart file is specified, it means that the number of runs indicated on the command is the additional number of runs.

  Topology or stimuli condition changes are allowed, but are not meaningful. The histogram and Monte Carlo statistics reported at the end of an incremental Monte Carlo "session" are computed using data from the previous session. The waveforms displayed versus the run index display the total information, that is, combine the successive runs.

Related options

CARLO_GAUSS (see "CARLO_GAUSS" on page 975), **SIGTAIL** (see "SIGTAIL=VAL" on page 982), **STATISTICAL** (see "STATISTICAL=0|1" on page 982), **DISPLAY_CARLO** (see "DISPLAY_CARLO" on page 999), **EXTMKSA** (see "EXTMKSA" on page 1000), **DUMP_MCINFO** (see "DUMP_MCINFO" on page 999)

Tolerance Setting Using DEV, DEVX or LOT

The size of the tolerance appears in a **.MODEL** command after the parameter keyword and value. The tolerance may be specified either as a percentage or an absolute quantity. To denote a percentage, the % sign must be used. Parametric expressions are allowed when specifying **DEV** or **LOT**.

The **DEV** tolerance parameter causes devices which use the same **.MODEL** statement to vary independently of each other, as illustrated in the following example:

```
c1 4 0 cmod 10p
c2 6 8 cmod 10p
.model cmod cap dev=10%
```

In the above example, both **c**1 and **c2** use the model cmod. Their nominal values are both 10pF. The **dev** declaration placed immediately after the **cap** keyword indicates that during a Monte Carlo analysis, their values may vary independently of each other by at most ± 10%. So, **c**1 could have a value of 9.9pF while **c**2 has the value of 10.1pF during a simulation run.

In order to use a Gaussian distribution, the DEV parameter is changed to DEV/GAUSS. Using the same example, changing the distribution to Gaussian would alter the **.MODEL** card as follows:

```
c1 4 0 cmod 10p
c2 6 8 cmod 10p
.model cmod cap dev/gauss=10%
```

Both **c**1 and **c2** may vary independently of each other with a Gaussian distribution, where the standard deviation is 10 percent of the nominal value, 10pF.

The **dev** tolerance is appropriate for situations where the variation of parameters is uncorrelated. The devices on a printed circuit board are such an example.

The **DEVX** specification forces Eldo to use a new random value for each instance of a subcircuit. The difference with **DEV** is that even if a parameter is used several times in the same subcircuit, only one value will be used for that particular instance.

Note

DEVX can only be used in a **.PARAM** statement and not in a **.MODEL** statement. It is impossible to have **DEV** and **DEVX** specified on the same parameter. If **DEV** and **DEVX** are specified on the same parameter, the last specification will be retained.

The **lot** tolerance setting causes devices which use the same **.MODEL** statement to vary with each other, as illustrated in the following example:

```
c1 4 0 cmod 10p
c2 6 8 cmod 10p
.model cmod cap lot=10%
```

This is the same as the previous example with the exception that the tolerance has been changed from **dev** to **lot**. Now **c**1 and **c**2 will always have the same value. They may be both equal to 9.9pF during one run and 10.1pF during another run, but **c**1 will not have a value of 9.9pF while **c**2 has the value of 10.1pF during the same run.

The **lot** tolerance is appropriate for situations where there is a variation of the parameters track. Devices in an integrated circuit are such examples.

Multiple Runs

It is necessary to specify the number of simulations to be run in the implementation of a Monte Carlo test. The computation time increases linearly with this number of simulations. The number of runs are specified in one of the parameters of the **.MC** command. Multiple runs of the selected analysis are carried out whereby the first one uses nominal component values, with subsequent runs varying model parameters according to the specifications given via the **LOT** and **DEV** tolerances on each **.MODEL** parameter.

Examples

```
.model rmod res dev=2%
...
.mc 5 v(n5)
.tran 1n 100n
.plot tran v(n5)
```

Specifies five Monte Carlo runs of the transient analysis at the output node n5. All devices with the model name rmod have a **dev** tolerance attached to their nominal value.

```
.model cmod cap lot/gauss=2%
...
.mc 5 v(n5)
.tran 1n 50n
.plot tran v(n5)
```

Specifies five Monte Carlo runs of the transient analysis at the output node n5. All devices with the model name cmod have a **lot** tolerance with a Gaussian distribution, where the standard deviation is two percent of the nominal value.

```
*MODEL definition
.model mod1 nmos niv=6 eox=25.0n mu0=600
+ nb=2.0e+16 kl=2.24u lot=0.5% gw=3.91u
+ gl=0.7u dev=0.07e-6 rec=0.15u vt0=0.55
...
.mc 7 vdb(n6)
.ac dec 10 1.0e3 1.0e9
.plot ac vdb(n6) vp(n6)
```

Specifies seven Monte Carlo runs of the AC analysis at the output node n6. All devices with the model name mod1 have a **lot** tolerance attached to the parameter **kl** and a **dev** tolerance attached to the parameter **gl**.

An example of this type of analysis can also be found in “Tutorial #7—Non-inverting Amplifier” on page 1363.

```
*MODEL definition
.model mod1 nmos niv=6 eox=25.0n mu0=600
+ nb=2.0e+16 kl=2.24u gw=3.91u
+ gl=0.7u rec=0.15u vt0=0.55
...
.mcmod mod1 kl lot=0.5% gl dev=0.07e–6
.mc 7 vdb(n6)
```

Specifies the same Monte Carlo run as the previous example, but using the **.mcmod** command.

For more information, see “.MCMOD” on page 715.

DEV variation specified with **.MODEL** statements (or **.MCMOD**) can refer to dimensions of the current object directly, without any need to encapsulate models into subcircuits, for example `E(*,<instance_parameter_name>)`.

```
.MODEL M nmos VTH=1 DEV={sqrt(E(*,l)*(E(*,W))*1.0e-5}
M1 p1 p2 p3 p4 m w=10u l = 3u
```

The two lines above are equivalent to the five below:

```
.SUBCKT foo d g s b param: w=1u l=1u
.model m nmos VTH=1 DEV={sqrt(l*w)*1.0e-5}
M1 d g s b m w=w l=l
.ENDS
X1 p1 p2 p3 p4 foo w=10u l=3u
```

A full example to illustrate this is shown below:

```
* test DEVX
M1 D1 G1 0 B mos1 w=w l=l as=0 ad=0 ps=0 pd=0 m=1
M2 D2 G2 0 B mos1 w=w l=l as=0 ad=0 ps=0 pd=0 m=1
.MODEL mos1 nmos
+ level=53 version=3.24
+ u0=300 tox=5n
+ vth0=1 DEV={sqrt(E(*,l)*(E(*,w)))*1.0e-05}
.param l=5e-6 w=10e-6
.op
.MC 10
.option display_carlo EXTMKSA
```

At the end of a Monte Carlo simulation, Eldo prints in the *.chi* file statistical information for each **.EXTRACT** command. The example below shows the *.chi* file entry for a Monte Carlo analysis with the following **.EXTRACT** command:

```
.EXTRACT ac label=fc3db xycond(XAXIS, vdb(s)=yval(vdb(s),1.0e3)-3)

Distribution of FC3DB
```

```
Range [ 8.7876E+04  1.2188E+06]
Nominal value:  8.5830E+04
Average value:  5.3650E+05
Standard Deviation:  3.9332E+05
Standard Deviation based on nominal run:  6.5967E+05
```

Two standard deviation results are provided: “standard deviation” is the RMS value of the deviation of output with respect to the average value; “standard deviation based on nominal run” is the RMS value of the deviation of output with respect to the nominal run.

The example below shows how **SIGBIN** can be specified to truncate the histogram to a certain number of sigmas.

```
* analyze main distribution when there are atypical values
v1 1 0 val
.param val=1 dev/trimodal=0.00002
.distrib trimodal
+ (-1     0) (-0.999 0.1) (-0.998 0)
+ (-0.1   0) ( 0     1  ) ( 0.1   0)
+ ( 0.998 0) ( 0.999 0.1) ( 1     0)
.extract dc label=v1 v(1)
.extract dc label=v1_norm '(v(1)-1.0)*1'
.dc
*
.mc 2000 print_extract=all  sigbin=4
.end
```

Specifying **SIGBIN** a value of 4 means atypical values are ignored for computing the min/max for the histogram. Eldo will gather all the samples above “mean$+$SIGBIN$\times$sigma” in the Above bin, and all the samples below “mean$-$SIGBIN $\times$sigma” in the Below bin.

.MCMOD

LOT & DEV Variation Specification on Model Parameters (Monte Carlo)

```
.MCMOD MNAME [(list_of_instances)] PAR LOT|DEV=VAL {PAR LOT|DEV=VAL}
.MCMOD MNAME PAR LOTGROUP=my_lot_group
```

This command is used to specify the amount of variation on a given model parameter using the **LOT** and/or **DEV** parameters. By specifying the variation in this way, no modification of the **.model** definition concerned is required. This command is used in combination with Monte Carlo and Worst Case analyses.

Note

If you specify a Lot/Dev variation in a **.MCMOD** file, the variation will be around whatever value is input for that model. This may seem obvious but the point here is that the worst case models might be used and the Monte Carlo analysis be run around that. Engineers not clear on "models" might do this without realizing.

Different entities are able to share the same distribution. Anywhere Eldo accepts **LOT**/**DEV** specifications, you can specify **LOTGROUP**=group_name.

.MCMOD expects the name of a **.MODEL** command as the first argument. However, when binning models are used, there are usually several models with the same prefix (**.MODEL** MYMOD.1... , **.MODEL** MYMOD.2...) and none matching exactly the name specified in the **.MCMOD** command (**.MCMOD** MYMOD). Consequently, **.MCMOD** accepts only the prefix name as the argument.

If a parameter is not a primitive, but depends on parameters with no LOT/DEV specification, then a LOT /DEV specification can be set on that parameter.

Parameters

- MNAME

  Model name.

- PAR

  Parameter to be varied by **LOT**, **DEV** or **LOTGROUP**.

- VAL

  Value of PAR **LOT|DEV**.

- list_of_instances

  List of instances; the subsequent parameter list and variation will only apply to the specified devices. The list should be separated by spaces, and enclosed in brackets, for example:

  ```
  .mcmod mod1 (m1 m2 m3) vt0=...
  ```

- **LOTGROUP**=my_lot_group

  Specifies a "lotgroup" to enable different entities to share the same distribution. Anywhere Eldo accepts **LOT**/**DEV** specifications, you can specify **LOTGROUP**=my_lot_group.

Please refer to ".LOTGROUP" on page 701 for more information.

Examples

```
*MODEL definition1
.model mod1 nmos niv=6 eox=25.0n mu0=600
+ nb=2.0e+16 kl=2.24u gw=3.91u
+ gl=0.7u rec=0.15u vt0=0.55
...
.mcmod mod1 kl lot=0.5% gl dev=0.07e–6
.mc 7 vdb(n6)

*MODEL definition2
.model mod1 nmos niv=6 eox=25.0n mu0=600
+ nb=2.0e+16 kl=2.24u lot=0.5% gw=3.91u
+ gl=0.7u dev=0.07e–6 rec=0.15u vt0=0.55
...
.mc 7 vdb(n6)
```

The above two netlist samples produce the same results.

```
.MODEL R R rho=25
.MCMOD R rho=20%
.MCMOD R r=20%
```

The example above shows the limitation of the command for R, L, and C models. The 2nd line is not valid and an error message is issued. The 3rd line is a valid command, parameter name is R.

```
.MODEL NMOS.1 VT0=... WMIN=1u WMAX=10u LMIN=1u LMAX=10U
.MODEL NMOS.2 VT0=... WMIN=11u WMAX=100u LMIN=1u
+ LMAX=10U
M1 ... NMOS W=9U l=9u
M2 ... NMOS W=50u l=9u
.MCMOD NMOS VT0 lot=10%
.option modwl
.end
```

In the above example, model NMOS.1 will be attached to M1, model NMOS.2 will be attached to M2. The **.MCMOD** command will apply to both NMOS.1 and NMOS.2, that is, the same random number will be used for NMOS.1 and NMOS.2.

```
.param p1 = 1
.param p2 = p1 lot = 5%
.param p3 = p2 dev = 5%
```

In the above example, MC variation on parameter p2 is accepted, MC variation on parameter p3 is ignored because its value depends on the value of parameter p2. The nominal value of p3 will be p2.

.MEAS

Measure Waveform Characteristics

```
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name
+ TRIG trig_spec TARG targ_spec
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name WHEN when_spec AT val
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name FIND wave WHEN when_spec
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name FIND wave AT val
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name FIND W('wave')
+ WHEN when_spec [FROM=val] [TO=val]
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name DERIVATIVE wave
+ WHEN when_spec
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name DERIVATIVE wave AT val
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name meas_k wave
+ [FROM=val] [TO=val]
.MEAS [ANALYSIS_INFO] [VECT] [CATVECT] label_name PARAM='expression'
```

This command can be used as an alternative to the **.EXTRACT** command. **.MEAS** has the same capabilities as **.EXTRACT**, however the syntax of **.MEAS** is occasionally preferred by some users. **.MEAS** can be used in three ways:

- Measuring a time interval or wave values
  Measurement starts when the **TRIG** conditions, as defined by the `trig_spec` parameters, are matched.
- Measurement on a wave
- Combination of measurements

For compatibility with other simulators, no asterisk * character is printed in the ASCII output (*.chi*) file before the **.MEAS** result.

As a comparison between **.MEAS** and **.EXTRACT**: **.EXTRACT** offers greater flexibility, however it requires that each entity appearing in a **.EXTRACT** must be saved in memory. **.MEAS** does not make this requirement.

The results are listed to the ASCII output (*.chi*) file. By default, measurement results are saved inside the *EXT* folder in the main *.wdb* file which can be read by EZwave. The command also creates a *.meas.wdb* file with the measurement results when option **MEASFILE** is specified in the netlist. If using the *.cou* format, the command also creates a *.meas* file when option **MEASFILE** is specified in the netlist. The *.meas* or *.meas.wdb* file will not always be created as it depends on the type of simulation and the specification of the **.MEAS** command.

Parameters

- **ANALYSIS_INFO**

  should be replaced with one of the following parameters;

  AC
  Measurement during AC analysis.

DC
Measurement during DC analysis.

DCTRAN
Measurement after the DC analysis performed prior to a TRAN analysis.

DCAC
Measurement after the DC analysis performed prior to an AC analysis.

DCSWEEP
DCSWEEP measurement.

TRAN
Measurement during TRAN analysis.

NOISETRAN
Measurement during NOISETRAN analysis.

- **VECT**

  By default, **.MEAS** returns the first value which matches the expression. But if keyword **VECT** is set on the **.MEAS** statement, then all values will be returned.

- **CATVECT**

  Works in the same way as **VECT** but in addition all measurements corresponding to all analyses (**.STEP/.TEMP**) will be combined. This functionality is usually used in conjunction with the **CONTOUR** function in the **.PLOT** command.

For more information on the **CONTOUR** function, see "CONTOUR" on page 798.

- label_name

  Identifies the **.MEAS** command in all output files. label_name must be defined.

- **TRIG**

  Measurement starts when the **TRIG** conditions, as defined by the trig_spec parameters below, are matched:

- trig_spec:

  WAVE **VAL**=val [**TD**=val] [**CROSS**=index|**LAST**] [**RISE**=index|**LAST**]
  + [**FALL**=index|**LAST**] [**SIG_H**=val] [**SIG_L**=val]
  or:

  AT[=]val

WAVE
Name of wave.

VAL
Threshold value.

TD
Time delay until measurement commences.

CROSS
Number of times the threshold must be crossed (in whichever direction) before measurement starts. Keyword **LAST** can be specified in place of a value.

RISE
Number of times the wave values rise above the threshold before measurement starts. Keyword **LAST** can be specified in place of a value.

FALL
Number of times the wave values fall below the threshold before measurement starts. Keyword **LAST** can be specified in place of a value.

SIG_H, **SIG_L**
High and Low signal values respectively. Default high and low values are taken from the threshold value **VAL**. These high and low values are used to validate a transition before incrementing the cross, rise or fall counter.

- **AT**

  Measurement starts/stops at `val`. The `val` depends on the analysis type: time for TRAN analysis, frequency for AC analysis, or the parameter (x-axis value) for DC analysis.

- **TARG**

  Measurement stops when the **TARG** conditions, as defined by the `targ_spec` parameters, are matched.

- `targ_spec`

  The arguments are the same as for `trig_spec`. If the time delay **TD** is not specified, the time delay specified in `trig_spec` will be used.

- **WHEN**

  Measurement stops when the **WHEN** conditions, as defined by the `when_spec` parameters, are matched.

- `when_spec`

  The arguments are the same as for `targ_spec`, except that both `WAVE` and `VAL` specifications are replaced by either of the following:

  ```
  WAVE=VAL
  WAVE1=WAVE2
  ```

- **DERIVATIVE**

  Keyword which can be used in place of **FIND**, to specify that it is the derivative of the `wave` that must be returned when **WHEN** specification is matched.

- **meas_k**

  Used for measurement on a wave. This can be one of the following keywords:

  AVG
  Average value of the waveform in the range `[FROM,TO]`.

RMS
RMS value of the waveform in the range [FROM,TO].

MIN
Minimum value of the waveform in the range [FROM,TO].

MAX
Maximum value of the waveform in the range [FROM,TO].

PP
Peak-to-Peak value of the waveform in the range [FROM,TO].

- PARAM='expression'

  Regular expression that combines the label names of the **.MEAS** commands. expression cannot be a wave name. Plot quantities such as i(), v(), lv9() are only accepted in a **.MEAS DC** analysis. For example:

```
.MEAS DC vth0 param='lv9(m)'
```

Examples

The example below will return the last time that **v**(in1) crossed value 2.0 while falling:

```
.MEAS TRAN mymo TRIG AT=0 TARG v(in1) val=2 FALL=LAST
```

The example below will return the value of v(1) when v(2) becomes equal to the v(5) measurement, starting after a delay of 3n:

```
.MEAS TRAN foo FIND v(1) WHEN v(2)=v(5) TD=3n
```

The example below will return the average of v(in1) between 3n and 5n:

```
.MEAS TRAN my_avg avg AVG v(in1) FROM=3n TO=5n
```

The example below will return the average of V(s):

```
.PARAM p1=1
.MEAS TRAN avg_name AVG V(s)
```

The example below will return the value of measurement named avg_name + value of parameter P1:

```
.MEAS TRAN f2 PARAM='avg_name+p1'
```

The following example shows how the **VECT** keyword can be used to return all the values which match the expression. By default, **.MEAS** returns the first value which matches the expression.

```
v1 1 0 pulse(0 5 0 1n 1n 8n 20n)
r1 1 0 1
.extract vect xup(v(1),2.5)
.meas vect tran memes trig at 0 targ v(1) val=2.5 rise=1
.tran 1n 100n
.end
```

The following example shows how the `W` notation can be used with the **`.MEAS`** command. This will search the value of wave `vp(1)` when wave `vdb(1)` will cross zero between 10 Hz and 1000 Hz.

```
.MEAS AC ph1 FIND W('vp(1)') WHEN vdb(1)=0 FROM=10 TO=1000
```

Related option

`FROM_TO` (see "`FROM_TO=0|1`" on page 964)

.MODDUP

Aspire/SimPilot Command

```
.MODDUP device_name [... device_name]
.MODDUP element_name
```

This command is useful in connection with SimPilot/Aspire only. It tells Eldo that for each `device_name` a private **.MODEL** command be created. Then, it will be possible for the SimPilot/Aspire engine to alter the content of this new model command via commands.

When the netlist is sourced, SimPilot/Aspire stores the **.MODDUP** arguments (device instance names) and relates these to the associated **.MODEL** commands. Therefore, the user does not need to manually duplicate **.MODEL** commands inside the netlist. These copies reside inside the Eldo simulator.

With `element_name` specified, the effect of this command is that the model attached to `element_name` is duplicated, and becomes private to that element. Parameters of the associated **.MODEL** command can be modified/displayed in interactive mode via the commands:

```
SET EM(Element_name,Model_parameter) = value
PRINT EM(Element_name,Model_parameter)
```

- `device_name`

  Device instance name.

- `element_name`

  Element name.

.MODDUP ‘carries’ parameter dependencies, for example:

```
.PARAM P1 = 1.0
.MODEL FOO NMOS VT0 = P1
M1 .... FOO ..
M2 .... FOO ..
.MODDUP M1
.STEP PARAM P1 1 2 1
...
.END
```

Here, **.STEP** `P1` will update the `VT0` parameter of `FOO` as well as updating the `VT0` parameter of the newly created model attached to `M1`.

For an example, please see Example—device mismatch for Eldo v4.4.1 or later of the *Aspire User’s Manual*.

.MODEL

Device Model Description

```
.MODEL MNAME TYPE NONOISE [PAR=VAL]
.MODEL LIB FILENAME MODNAME [LIBTYPE]
```

This command groups sets of pre-defined parameters which may be used by one or more devices. Models may be described directly in the input file or may be read from a library file using either the second syntax above or the **.LIB** and **.ADDLIB** commands. The second syntax allows no continuation lines.

It is also possible to specify model parameters on the instance command. The effect is that a private model will be created for that instance.

By default, when Eldo encounters multiple definitions of **.MODEL** statements an error is reported and the simulation is stopped. Specify option **USEFIRSTDEF** to force Eldo to only use the first definition (any further definitions are ignored) to allow the simulation to proceed.

Parameters

- MNAME

  The model name. It must not start with a number.

- TYPE

  Defines the model used. The following models are available:

Table 10-16. Model Types

RES	Resistor
R	RC wire
CAP	Capacitor
IND	Inductor
NPN	NPN bipolar junction transistor
PNP	PNP bipolar junction transistor
LPNP	Lateral PNP bipolar junction transistor
D	Diode
NMOS	N-channel metal oxide field effect transistor
PMOS	P-channel metal oxide field effect transistor
NJF	N-channel junction field effect transistor
PJF	P-channel junction field effect transistor
NSW	N-type switch (SC)

Table 10-16. Model Types

PSW	P-type switch (SC)
OPA	Operational amplifier (SC)
MODFAS	Analog macromodel (including LDTL Lossy Transmission Line)
SP	Sampled Matrix model (Lossy Transmission Line)
U	U model (Lossy Transmission Line)
W	RLGC model (Lossy Transmission Line)
LOGIC	Digital Gate
A2D	Analog-to-Digital converter
D2A	Digital-to-Analog converter
MACRO	Macromodel (including S parameter block)

- **LIB**

  Keyword indicating a model library file is to be used.

- FILENAME

  Name of the library file that contains the model description.

- MODNAME

  Name of the model stored in library file FILENAME. See the "Library variant management" on page 694 in the **.LIB** command description for details.

- **NONOISE**

  Specifies that no noise model will be used for the corresponding object when performing noise analysis. Therefore, the object presents no noise contribution to the noise analysis. This can be overwritten on an instance specification using **NOISE**=1.

- PAR=VAL

  Name and value of a model parameter. Model parameters not given values are assigned default values. Model parameters may also be declared via the **.PARAM** command. See the ".PARAM" on page 778 for further details.

  The parameter **BULK**=node_name can be specified on a **.MODEL** NMOS|PMOS command. This will connect the bulk node to node_name if it is not specified in the instantiation of the model. By default node_name is 0.

- LIBTYPE

  Name of a library variant to be used.

For more information on electrical parameters of models, refer to the Device Models chapter.

Related option

USEFIRSTDEF (see "USEFIRSTDEF" on page 962)

Examples

```
*MODEL definition
.model rmodel res tc1=0.001 tc2=0.005
...
*main circuit
r2 n1 n19 rmodel 2.5k
```

Specifies the resistor `r2` of model type `rmodel`.

```
*BJT model definition
.model qmod npn bf=160 rb=100 cjs=2p
+ tf=0.3n tr=6n cje=3p cjc=2p vaf=100
...
*main circuit
q23 10 24 13 qmod
```

Specifies the bipolar transistor `q23` of model type `qmod`.

```
*OPAMP model definition
.model ampop modfas voff=100e-6
...
*main circuit
yopa1 opamp1 n2 n1 n3 0 model: ampop
```

Specifies the single-stage 1-pole op-amp `yopa1` with the electrical parameters specified in the model `ampop`.

```
*NOR# .MODEL definition
.model nor_1 logic vhi=5 vlo=-5 vth=0
+ tpd=2.5n cin=0.5p
...
*main circuit
nor#_1 n1 n2 n3 n4 o1 nor_1
```

Specifies a NOR gate **nor#_1** with four input nodes `n1`, `n2`, `n3` and `n4` and an output node `o1`, the parameters of which are described in the model `nor_1`.

```
M1 ... MOD1 w=1u l=1u M(DW)=0.5u
M2 ... MOD1 w=1u l=1u
.model MOD1 NMOS DW=3u LOT=3%
```

In this example, a private model will be created for `M1`. Changing the `DW` parameter value of model `MOD1` (via **.STEP** for instance) will not affect the value of parameter `DW` for the model attached to device `M1`.

Note

This feature only works for MOS, BJT, Diodes and JFET. Care must be taken when using this feature, since a private model is created for each instance. The memory requirement on circuits that contain many such model parameter specifications on instance commands can be very high. Monte Carlo specifications are not propagated to the new model. In the example above, the `DW` value of instance `M1` will not be changed during a Monte Carlo analysis.

In the following example the bulk node will be connected to node 4. This is because a node has not been specified in the instantiation of the MOS model and the parameter BULK *has* been specified in the **.MODEL** command.

```
M1 1 2 3 N W=10u l = 3U
.MODEL N NMOS BULK = 4
```

If **BULK** was omitted, by default the bulk node would be connected to node 0 (ground node).

```
.model res r r=2 NONOISE
r1 1 2 res 1k noise = 1
r2 2 3 res 1k
```

This example shows use of the **NONOISE** parameter. In this example, device r1 will generate noise, not r2.

Monte Carlo and Models

The Monte Carlo command is used in combination with extra parameters placed in the **.MODEL** command. Each **.MODEL** parameter to be subjected to statistical variation, may have two extra related parameters added, **DEV** and **LOT**. The significance of these parameters is explained in the following paragraphs.

An example of a **.MODEL** command, modified to include Monte Carlo analysis parameters, is shown below:

```
.model mmod nmos vto=0.65v dev=0.4v
+ tox=1.5e-7 dev=0.2e-7 lot=5%
```

The first **dev** declaration refers to the **vto** parameter, and the second (combined) **dev** & **lot** declaration refers to the **tox** parameter.

.MODLOGIC

Digital Model Definition

Caution

Obsolete Syntax!
Kept for compatibility reasons only and should no longer be used.

For the new syntax, see "Digital Model Definition" on page 449.

```
.MODLOGIC MNAME [VHI=VAL1] [VLO=VAL2] [VTH=VAL3] [VTHI=VAL4]
+ [VTLO=VAL5] [TPD=VAL6] [TPDUP=VAL7] [TPDOWN=VAL8]
+ [CIN=VAL9] [DRVL=VAL10] [DRVH=VAL11]
```

Used for the definition of digital gate models.

Parameters

For detailed information on the parameters, see "Digital Model Definition" on page 449.

.MONITOR

Monitor Simulation Steps

```
.MONITOR ANALYSIS [=] [modulo]
```

This command can be used to display the steps taken by the simulator when doing an AC, TRAN or DCSWEEP simulation. Current TIME/FREQ or DCSWEEP value is also displayed. This command can be used for debugging purposes, that is, to see how the simulation proceeds.

Eldo will display the current time and time step every modulo steps.

Parameters

- `ANALYSIS`

  Analysis type for which you request simulation steps to be monitored. Can be one of the following:

  DC
  Specifies that a DC analysis is monitored.

  AC
  Specifies that a AC analysis is monitored.

  TRAN
  Specifies that a transient analysis is monitored.

- `modulo`

  Number of steps at which the information is displayed. Default is zero, meaning all values are printed out.

Examples

```
.MONITOR AC
```

Eldo will display all the frequency points in an AC analysis.

.MPRUN

Multi-Processor Simulation

```
.MPRUN [ALL|HOST={host[(nbjobs)]}|FILE=filename}] [NBLICENSES=val]
+ [MAX_NBJOBS=val] [CLEAN=YES|NO] [QUEUE=YES|NO] [SETENV=YES|NO]
+ [VIEW_COMMAND=YES|NO] [CHECK_DELAY=val] [INIT_FILE=filename]
+ [DEFAULT_INIT=YES|NO] [NETWORK_DIR=directory]
+ [CD_WORKDIR=YES|NO] [LOGFILE=YES|NO]
+ [USE_LOCAL_HOST=YES|NO] [FLAT=YES|NO]
+ [FILE_PREFIX=(name1,name2,...,nameX)]
+ [USE_SSH=YES|NO] [SSH_OPTIONS="{<options>}"]
+ [CHECK_ALL_HOSTS=YES|NO]
+ [SYNCHRO_NOMINAL_MC=YES|NO]

.MPRUN DISPATCHER ...
```

See ".MPRUN and external dispatchers" on page 734 for **.MPRUN DISPATCHER** syntax.

This command is used to run multiple simulations on one multi-processor machine, or on many machines. **.ALTER**, **.MC**, **.TEMP**, **.STEP**, **.DATA** and **.OPTIMIZE** are distributed by this command. It can also be used with data sweeps on **.AC** and **.TRAN** commands.

Note

Eldo is also capable of multi-threading simulations to speed-up simulation. See "Multi-Threading Eldo Simulations" on page 58 for information.

This command uses a client/server architecture to ensure maximum efficiency. Once remote jobs have been submitted, the localhost may or may not start a simulation depending on the **USE_LOCAL_HOST** option. In both cases it has the responsibility to merge individual runs, unless the **FLAT** method is enabled. The message "Collecting results" is printed when the main job is waiting for results of a run.

Both main and remote jobs are not restricted to some runs. A side effect for very small simulations is that the localhost (with **USE_LOCAL_HOST**=**YES**) may perform all runs before remote jobs have established connections.

The child processes are by default launched through an *rsh* call which inherits the environment variables used in **.INCLUDE** or **.LIB** statements. Temporary results are stored in subdirectories named *<NETLIST_NAME>.part<X>* where *X* is an incrementing counter. By default—and unless they are in use—temporary directories are removed once the simulation is complete.

Child processes can be submitted either by *rsh*, *ssh*, or by an external dispatcher. These specifications are exclusive.

Notes

1. In all cases, Eldo will warn you if a host cannot be used. This will happen when:
   - o You do not have the permissions to see the machine and/or to write in the working directory. This error must be fixed by the system administrator before the **.MPRUN** command can be used.
2. The netlist must be in a shared directory (a place visible from the other machines on the network).
3. If using local installations of Eldo, the installation patches (not the binaries) must be strictly identical on all machines.
4. Users need to be able to rlogin to other machines without supplying password.

   If a user needs a password to perform an rlogin to other servers, when Eldo attempts to rlogin to other systems to launch tasks it will fail because the other machines require passwords and Eldo cannot supply them.
5. The result of a **.CALL_TCL** command used with **.MPRUN** is unpredictable since the behavior of the Tcl function is unknown to Eldo. It does not know if the command opens output files or returns a waveform.
6. **.MPRUN** cannot be specified inside a **.ALTER** section.

Parameters

Note

All *boolean* keywords, that is, **CLEAN**, **SETENV**, **QUEUE**, **VIEW_COMMAND**, **CD_WORKDIR**, **LOGFILE** can be set using: **<keyword>[=YES|NO|1|0]**. For example, specifying the **SETENV** keyword is equivalent to **SETENV=YES**, which is equivalent to **SETENV=1**. Additionally, when the **SETENV** keyword is not specified it will use its default value, in this case it would be **SETENV=NO**, which is equivalent to **SETENV=0**.

- **ALL**

  This is the default. This keyword specifies Eldo to run the simulation on all the processors of the machine. Eldo will find the number of processors of the machine and distribute the tasks between them. If the machine has one processor, Eldo will run the simulation normally without taking this command into consideration.

- **HOST**={host1[(nbproc1)] host2[(nbproc2)]... hostN[(nbprocN)]}

  Specifies Eldo to run the simulation on the list of machines: host1, host2, ... hostN (commas are optional). Eldo will distribute the tasks on the list of machines specified. nbjobs is an optional parameter that explicitly tells Eldo the maximum number of jobs that can be submitted on this machine. On multi-processor stations, this number should be the number of processors.

- **FILE**=filename

  Specifies the name of a file which contains a list of machine names (with a number of processors if needed). The file can have any extension or no extension at all. The first line of the file is read. Example file contents:

  ```
  pluton  kebra(3)
  morkai(2) nao
  cochise
  ```

- **NBLICENSES**=val

  Specifies the maximum number of licenses that this job can use. It must be greater than 1 to be taken into account, since the parent process always takes its own license.

- **MAX_NBJOBS**=val

  Specifies the maximum number of jobs that can be submitted for all machines.

- **CLEAN**[=YES|NO]

  Default value is YES. This specifies Eldo to remove temporary files created in any child process subdirectories, together with removing the subdirectories themselves. If keyword is set to NO, the temporary files are not removed.

- **QUEUE**[=YES|NO]

  Instructs the system to wait for the release of a license if one is not immediately available. A consequence is that the parent process will hang until its child process has finished. Default value is NO.

- **SETENV**[=YES|NO]

  Keyword specifies Eldo to reuse all environment variables in child processes, and not only these given on **.LIB** and/or **.INCLUDE** statements. Default value is NO.

- **VIEW_COMMAND**[=YES|NO]

  Prints the command Eldo submits to the remote host and the contents of the command file. Default value is NO.

- **CHECK_DELAY**=val

  Forces Eldo to wait val seconds while checking host connections. This is useful when using both Linux and Unix networks, since a delay can appear between the time when a remote command is executed on Unix and the time when the result of the command is effective on Linux.

- **INIT_FILE=**filename

  This allows you to define a script file which is sourced before running child processes.

- **DEFAULT_INIT**[=YES|NO]

  Default value is YES. If keyword is set to NO, it tells Eldo that the script specified in **INIT_FILE** will replace any other default. This should always be used in conjunction with **INIT_FILE=**filename. If using this, it is your responsibility to ensure that the script file correctly sets the path and required variables for Eldo to run.

- **NETWORK_DIR**=directory

  Specifies the network name of the directory where the netlist is located.

- **CD_WORKDIR**[=YES|NO]

  Forces Eldo to change the working directory in the remote environment to the directory where the netlist is located. Default value is YES.

- **LOGFILE**[=YES|NO]

  This controls the redirection of the standard output of sub-processes. The default value is YES, which means that Eldo dumps the standard output in *<NETLIST_NAME>.log* in the temporary directories.

- **USE_LOCAL_HOST**[=YES|NO]

  Setting this option to NO tells Eldo that it cannot use the local host to perform some simulations. Default value is YES (the local host is the machine on which the main process has been launched).

- **FILE_PREFIX**=(name1,name2,...,nameX)

  In the case of splitting **.ALTER** statements, you may want to rename the output files of each run. If a flag is given after **.ALTER**, it is used as the name. Output names can also be redefined with this **FILE_PREFIX** option. For example if a netlist contains two **.ALTER** statements and the **.MPRUN** option **FILE_PREFIX**=(first,second, third), output files for each run will be:

  - for the netlist before **.ALTER** statements: *first.chi*, *first.wdb*, and so on
  - for the netlist after the first **.ALTER** statement: *second.chi*, *second.wdb*
  - for the netlist after the second **.ALTER** statement: *third.chi*, *third.wdb*

- **FLAT**[=YES|NO]

  The usual **.MPRUN** mechanism uses temporary directories to store the results of intermediate runs. In the case of splitting **.ALTER** statements, if this option is set to YES, Eldo will use different names for each intermediate run instead of using directories. **.ALTER** sections are submitted as if they are independent netlists. It is recommended to use this option in conjunction with **FILE_PREFIX** to define the names of the runs. Default value is NO, and it is ignored if no **.ALTER** statements are declared.

- **USE_SSH**[=YES|NO] [**SSH_OPTIONS**="{<options>}"]

  Setting the **USE_SSH** option to YES tells Eldo to work with *ssh* (secure shell) instead of *rsh* (remote shell). Default value is NO (remote shell). Secure shell options can be specified with the **SSH_OPTIONS** parameter.

- **CHECK_ALL_HOSTS**[=YES|NO]

  If YES, Eldo will check that all hosts are valid and report warning or error messages depending on the host status. Default value is NO, which means the connection to all hosts specified with the HOST argument is not checked. In addition:

If **USE_LOCAL_HOST**=YES, none are checked.

If **USE_LOCAL_HOST**=NO, Eldo just checks that at least one connection is active. It reduces the overhead before starting the mprun simulation.

- **SYNCHRO_NOMINAL_MC**[=YES|NO]

  If YES, the **.MPRUN** mechanism will impose synchronization points at each nominal Monte Carlo run in order to provide the same DC solution to all remote processes. Default value is NO.

  During a Monte Carlo analysis, the DC solution of the first run is taken as a reference to speed-up convergence of the following runs. When **.MPRUN** is used, the first run performed by each individual process is not the same. This means the reference DC solution is not uniform over all processes. The visible effect may be very small variations on waveforms and extracts (numerical noise).

  If the design contains several levels of multi-run commands, including a **.MC**, the **.MPRUN** can significantly slow because all processes will have to wait until the nominal run is finished.

Usage

Before performing a run on a remote host, the **.MPRUN** command must initialize the remote environment. By default, Eldo does the following before running a simulation:

```
cd <working_directory>
setenv MGC_AMS_HOME ...
setenv LM_LICENSE_FILE ...
```

If you specify **INIT_FILE=**<filename>, the sequence becomes:

```
cd <working_directory>
source <filename>
setenv MGC_AMS_HOME ...
setenv LM_LICENSE_FILE ...
```

and if you also specify **DEFAULT_INIT=**NO, the sequence becomes:

```
cd <working_directory>
source <filename>
```

and if you also specify **CD_WORKDIR**=NO, the sequence becomes:

```
source <filename>
```

and if you remove the **INIT_FILE**=<filename>, absolutely no initializations will be performed before Eldo is run.

Examples

```
.MPRUN HOST=host1, host2 NBLICENSES=2 QUEUE=YES
```

Specifies Eldo to run the simulation on both the host1 and host2 machines. A maximum of two licenses can be used by this simulation. If a license is not immediately available, the system will wait for the release of a license.

```
.MPRUN HOST=host3(2) host4
+ INIT_FILE=script.shell DEFAULT_INIT=NO
```

Specifies Eldo to run the simulation on both the host3 and host4 machines, also with two processors of host3 specified as running the simulation. A script file *script.shell* will be sourced before running child processes. This script will replace any other default initialization file.

An example standard output of an **.MPRUN** job is as follows:

```
.TEMP -10 10
.MPRUN

Memory space allocated (bytes): 3603249
22 elements
16 nodes
3 input signals

(localhost) - No limitation applied: a maximum of 1 job(s) will be
submitted.
(localhost) - Job Id 0 submitted to host SHAMBHALA
(localhost) - Standard output of localhost redirected to file:
              test02.main.log
(localhost) - Starting run 1.
(localhost) - Completed run 1 of 2
(localhost) - Collecting results of run 2
(shambhala - 0) - Starting run 2.
(shambhala - 0) - Completed run 2 of 2

***>GLOBAL CPU TIME 0s 220ms <***

***>GLOBAL ELAPSED TIME 6s <***
```

.MPRUN and external dispatchers

```
.MPRUN DISPATCHER= [LSF |
+ (dispatcher_name, install_check_cmd, submission_cmd]
+ [REMOVE_QUOTE=YES|NO] [DISPATCHER_OPTIONS=options]
+ [NBLICENSES=val] [MAX_NBJOBS=val] [CLEAN=YES|NO]
+ [QUEUE] [SETENV] [VIEW_COMMAND] [CHECK_DELAY=val]
+ [INIT_FILE=filename [DEFAULT_INIT=YES|NO]]
+ [USE_LOCAL_HOST=YES|NO] [FLAT=YES|NO]
+ [FILE_PREFIX=(name1,name2,...,nameX)]
+ [LSF_JOB_PREFIX=lsf_prefix]

.MPRUN
+ DISPATCHER_TEMPLATE=command_line
+ [USE_LOCAL_HOST=YES|NO] [FLAT=YES|NO]
+ [FILE_PREFIX=(name1,name2,...,nameX)]
```

A dispatcher is software which shares and manages computer resources across a network. Instead of a basic remote shell command, the dispatcher uses various criteria (memory usage, CPU charge) to determine which machine is suitable to run the simulation on. The second syntax above is useful for **.ALTER** dispatching only. For all other options listed above, please refer to the descriptions on the previous pages.

- **DISPATCHER=LSF**

  Specifies Eldo to use LSF as external dispatcher. This syntax is a shortcut to **DISPATCHER**=("LSF","bsub -V","bsub").

For more information on LSF, please visit the Platform web site: http://www.platform.com

- **DISPATCHER**=(dispatcher_name, install_check_cmd, submission_cmd)

  The general syntax to configure a dispatcher.

  dispatcher_name

  name of the software (required for print purpose only).

  install_check_cmd

  a command that can prove the software is accessible.

  submission_cmd

  the command that submits a job to the engine.

- **REMOVE_QUOTE**[=YES|NO]

  Modifies how Eldo manages double quotes inside the **DISPATCHER_OPTIONS** argument. If defined, it is mandatory to set it before the **DISPATCHER_OPTIONS** argument. Default value for LSF is YES, and NO for other dispatchers.

- **DISPATCHER_OPTIONS=options**

  This keyword allows to send extra options to the dispatcher submission command. The dispatcher options must be enclosed in ( ), { }, " " or ' ‘. If enclosed in ( ) or { } Eldo may add some spaces between items. To keep the exact syntax, use " " or ' ' if the options contains some double quotes.

- **DISPATCHER_TEMPLATE**

  This option allows to completely override the usual mprun mechanism. It is useful for **.ALTER** dispatching only. The user is in charge of providing a valid shell command which will dispatch jobs. This command can use predefined variables which are substituted by Eldo before executing the command. These variables are:

  %NETLIST_NAME%
  stands for the name of the netlist %RUN_NAME% stands for the name of each run (may be redefined with FILE_PREFIX)

  %RUN_NUMBER%
  stands for an absolute run counter (starting from 1) %MPRUN_OPTIONS% is a mandatory variable which represents internal options added by Eldo.

- **LSF_JOB_PREFIX=**`lsf_prefix`

  Specifies prefix for dispatched job names. By default, jobs submitted to LSF are named *<netlist_base_name>.part<job_id>*. If this argument is specified, the submitted jobs will be named *<lsf_prefix_name>.part<job_id>*.

Example with LSF

```
.MPRUN DISPATCHER=LSF
+ DISPATCHER_OPTIONS=(-q myqueue -m "host1 host2")
+ MAX_NBJOBS=5
```

Specifies Eldo to use the LSF management system as a dispatcher for the remote jobs. The simulation will be run on both the host1 and host2 machines. A maximum of five jobs will be submitted to LSF. `-q myqueue` is the LSF option which controls the Batch Queue to which the jobs will be submitted.

Example with .ALTER dispatching

Suppose file *mpex.cir* contains the following command in conjunction with two **.ALTER** statements:

```
.mprun
+ flat=yes
+ use_local_host=no
+ dispatcher_template="eldo %NETLIST_NAME% -queue %MPRUN_OPTIONS% >
%RUN_NAME%.log"
+ file_prefix=(first, second, third)
```

After parsing the netlist, Eldo will immediately execute the following commands:

```
eldo mpex.cir -queue ... -out first > first.log eldo mpex.cir -queue ...
-out second > second.log eldo mpex.cir -queue ... -out third > third.log
```

This example only demonstrates the syntax since it does no actual dispatching.

.MSELECT

Automatic Model Selection

```
.MSEL[LECT] dummy [MODELS] mod1 [mod2 [mod3 [...]]]
```

This command allows you to select models automatically for MOS devices. The selection is based on:

- the size and temperature of the specific device (W, L, TEMP)
- the size and temperature constraints of each model in the list provided (WMIN, WMAX, LMIN, LMAX, TEMPMIN, TEMPMAX)

It is not allowed to have a model statement with the same name as an mselect dummy model name. If a model statement has the same name as an mselect dummy model name, Eldo will display an error message, for example:

```
ERROR  953:Dummy model name MOD1 on .MSELECT statement is also defined on
a .MODEL statement
```

Parameters

- `dummy`

  Dummy model name that is used on the devices for which you want automatic model selection.

- `MODELS`

  Optional keyword used only to enhance **`.MSELECT`** statement readability.

- `mod1 ... modn`

  List of model names from which a new model is selected.

Additional information

- Device temperature can also be specified. Eldo will check against model parameters TEMPMIN/TEMPMAX to select the right device
- If a value is not specified for any of the models parameters (TEMPMIN/TEMPMAX/LMIN/LMAX/WMIN/WMAX) the checks versus the limits are not done.
- Models are searched in the order there are given on the mselect statement
- Many mselect with the same name can be defined. The previous definitions are automatically overwritten. Eldo will use the last one, for example:

  ```
  .mselect dummy2  models mo1 mo2.
  .mselect dummy2  models  mo1 mo2 mo3 mo4
  ```

  The last one will be used.

- When none of the models defined on a mselect fit with the device parameter an error message is displayed, for example:

```
ERROR  845: OBJECT "M1": None of the models in .MSELECT  fits this
instance
```

- If the one the models specified on the mselect model list does not exist a simple warning is emitted, for example:

```
Warning 488: COMMAND .MSELECT:  Model MOD7 is not defined
```

- When 2 different types of models (example NMOS and PMOS)   an error is emitted, for example:

```
ERROR  945: Wrong Model type for MOD2 on .MSELECT statement - mixing
models type is not allowed
```

Limitation

Eldo implementation of mselect only work for MOS devices and models.

Example

An example with mselect is below.

```
.MODEL mod11 ...
.MODEL mod12 ...
.MODEL mod21 ...
.MODEL mod22 ...

.mselect mod2 MODELS mod21  mod22
.mselect mod1 MODELS mod11  mod12

M1 A G VDD VDD MOD2 W=120U L=5.5U
M2 B G VDD VDD MOD2 W=120U L=5.5U
M3 D K A VDD MOD2 W=116U L=3.5U
M4 S K B VDD MOD2 W=116U L=3.5U
M5 C I VSS VSS MOD1 W=63U L=6U
M6 A EP C VSS MOD1 W=130U L=4U
M7 B EN C VSS MOD1 W=130U L=4U
M8 D D FF VSS MOD1 W=5.5U L=4.5U
M9 S D E VSS MOD1 W=5.5U L=4.5U
M10 FF E VSS VSS MOD1 W=42U L=4U
M11 E E VSS VSS MOD1 W=42U L=4U
M12 G G VDD VDD MOD2 W=14.5U L=5.5U
M13 G G H VSS MOD1 W=9U L=5.5U
M14 I I H VDD MOD2 W=19U L=4.5U
M15 I I VSS VSS MOD1 W=6U L=6U
M16 J G VDD VDD MOD2 W=20U L=5.5U
M17 J J K VSS MOD1 W=26U L=3.5U
M18 NL I K VDD MOD2 W=3U L=3.5U
M19 NL NL VSS VSS MOD1 W=4U L=3.5U
```

There are two mselect statements and 4 models definitions. The MOS devices are instantiated using the mselect dummy names instead of models real names. In this example only W and L instance parameters are used. The actual models used on the instances are selected when these dimensions are within the model parameters LMIN/LMAX and WMIN/WMAX

As the results of the selection you will have:

```
* M1, M2 M3 M4                        mapped to MOD21
* M12 M14 M16 M18                     mapped to MOD22
* M5 M8 M9 M10 M11 M13 M15 M17 M19   mapped to MOD11
* M6 M7                               mapped to MOD12
```

.NET

Network Analysis

2-port network

```
.NET output input RIN=val ROUT=val
```

1-port network

```
.NET input RIN=val
```

The **.NET** command is another approach to extract the S parameters (Scattering parameters), the Y parameters (Admittance), the Z parameters (Impedance) or the H parameters (Hybrid) in the frequency domain for a specified circuit.

The circuit can only have one or two ports.

Parameters

- `input`

  Can be V source or I source.

- `output`

  Can be V source, I source or V(NP,NN).

- `RIN ROUT`

  Specify the values of access resistors of the input and output port.

Example

```
.ac dec 500 1e6 10e6
.net v(outputnode) vinput_source RIN=50 ROUT=50
```

The following example shows a 1-port network extraction on voltage source `v1` with input access resistance of 50 ohms.

```
v1 1 0 ac 1 0
r1 1 2 1
c1 2 0 10p
.ac dec 10 1 10G
.plot ac sdb(1,1)
.plot ac sp(1,1)
.net v1 rin=50
```

.NEWPAGE

Control Page Layout

```
.NEWPAGE
```

This command allows the control of the saved windows file in the EZwave waveform viewer. It allows lines to be inserted into an Eldo netlist, with the effect that **.PLOT** commands located between two **.NEWPAGE** commands will be plotted in the same EZwave window. **.NEWPAGE** only acts on *.wdb* and *.cou* files.

Note

The number of items per **.PLOT** is not limited. It is possible to have any number of waves in the same plot, although reading the ASCII plot may be difficult.

See ".PLOT" on page 791 for more information.

Example

```
.PLOT TRAN (v1) (v2)
.PLOT TRAN (v3) (v4)
...
.PLOT TRAN (v15)
.NEWPAGE
.PLOT TRAN (v15)
...
```

This allows the user to control page layout, Eldo will plot the graphs following the **.NEWPAGE** command in a new EZwave window.

.NOCOM

Suppress Comment Lines from Output File

```
.NOCOM
```

This command suppresses any comment lines in the ASCII output (*.chi*) file which come after it in the netlist. Saves disk space.

Example

```
example title
.nocom
*This is a sample comment line
r1 1 2 5
*here is another
...
.end
```

.NODESET

DC Analysis Conditions

```
.NODESET V(NN)=VAL [SUBCKT=subckt_name] {V(NN)=VAL [SUBCKT=subckt_name]}
```

This command is used to help calculate the DC operating point by initializing selected nodes during the first DC operating point calculation. After the first calculation has been completed the node values are "released" and a second DC operating point calculation is started. This command is useful when the whereabouts of the DC operating point is known, enabling the simulator to converge directly to it and also for bistable circuits or circuits with more than one operating point.

The **.NODESET** command differs from the **.GUESS** command in so far as when using **.NODESET** node voltages are fixed for the duration of the first DC calculation, whereas the node voltages are only initialized for the first iteration of a DC operating point calculation when using **.GUESS**. It is very important to specify realistic **.NODESET** values as convergence problems may occur when this command is not used properly.

Note

By default, the first **.NODESET** specification has precedence over subsequent **.NODESET** specifications. Setting **.OPTION LICN**, the last **.NODESET** specification will have precedence.

See "LICN" on page 979 for further information.

Parameters

- **V(**NN**)**=VAL

  Voltage at node NN in volts.

- **SUBCKT**=subckt_name

  If specified it will fix the voltage of the preceding node in all instances of the subcircuit subckt_name.

Examples

```
.nodeset v(n4)=6v v(n5)=2v v(n6)=–5v
```

Specifies that during the first DC operating point calculation, the initial values for the voltages at the nodes n4, n5 and n6 be initialized to 6V, 2V and –5V respectively.

```
.nodeset v(2)=3v SUBCKT=sub1 v(4)=-2v SUBCKT=sub2
```

Specifies that during the first DC operating point calculation, the initial values for the voltages at node 2 of subcircuit sub1 and node 4 of subcircuit sub2 will be initialized to 3 and –2V respectively.

.NOISE

Noise Analysis

```
.NOISE OUTV INSRC NUMS
```

The **.NOISE** command controls the noise analysis of the circuit and must be used in conjunction with an AC analysis.

The results of noise analysis runs are output using the **.PRINT** and **.PLOT** commands.

It is possible to control the output of the noise information via the **NOXTABNOISE** option, see "NOXTABNOISE" on page 1005.

Note

NOISE analysis may also be run from within a **.TRAN** command, see "AC in the middle of a .TRAN" on page 543 for more details.

Parameters

- OUTV

  Name of the output voltage node for which the equivalent output noise is to be calculated. The syntax is as follows:

  V(N1[, N2]**)**
  Specifies the voltage difference between nodes N1 and N2. If N2 and the preceding comma are omitted, ground is assumed.

  I(Vxx**)**
  Specifies the first argument as a voltage source.

- INSRC

  Name of the input voltage or current source for which the equivalent input noise is to be calculated.

- NUMS

  Indicates that only every NUMth frequency point is stored for print-out. The contribution of every noise generator in the circuit is printed at every NUMth frequency point. If NUMS is zero, no print-out is made. NUMS can be specified as a parameter or as an expression.

Example

```
.ac dec 70 100k 10meg
.noise v(5) vin 70
...
*output control
.plot noise inoise onoise
.plot noise db(inoise) db(onoise)
```

Specifies `vin` as input noise reference and `v(5)` as the voltage at the summing point. The noise is averaged over seventy frequency points and the input and output results are to be plotted on the same graph, the limits of which are controlled by the `.AC` command.

An example of this type of analysis can be found in "Tutorials" on page 1341.

.NOISE_CORREL

Noise Source Correlation

```
.NOISE_CORREL VN1 VN2
+ {f k_r k_i}
```

Defines correlation coefficients between two independent noise sources.

Parameters

- `VN1, VN2`

  The two noise sources to be correlated.

- `f, k_r, k_i`

  `k_r` and `k_i` define the real and imaginary parts respectively of the correlation coefficient between the two noise sources `VN1` and `VN2` at frequency `f`.

Example

```
V1  1  0  four fund1  ma  (1)  1  -90
+   noise  table
+   100    1e-5
+   1k     1e-5
+   10k    1e-5

R1  1  3  1k

V2  2  0  dc  2
+   noise  table
+   100    1e-5
+   1k     1e-5
+   10k    1e-5

.noise_correl  V1   V2
*+ <f>  <k_r>   <k_i>
+  100  -1  0
+  1k   -1  0
+  10k  -1  0

R2  2  3  1k
R3  3  0  1k
```

.NOISETRAN

Transient Noise Analysis

```
.NOISETRAN FMIN=VAL FMAX=VAL NBRUN=VAL [NBF=VAL] [AMP=VAL]
+ [SEED=VAL] [NOMOD=VAL] [NONOM] [TSTART=VAL] [TSTOP=VAL]
+ [MRUN] [ALL] [NBBINS=VAL] [FMIN_FLICKER=VAL]
```

This command is used to control the transient noise analysis of a circuit and must be used in conjunction with a transient (**.TRAN**) analysis, and not with a Monte Carlo (**.MC**) analysis.

For further information, see the chapter on "Transient Noise Analysis" on page 1023.

It is possible to define the three parameters **FMIN**, **FMAX** and **NBF** for each noisy component; Resistor, Junction Diode, BJT—Bipolar Junction Transistor, JFET—Junction Field Effect Transistor, MESFET—Metal Semiconductor Field Effect Transistor, MOSFET, Independent Voltage Source and Independent Current Source. Please refer to the appropriate sections.

To reduce the CPU time, parallel noise runs are performed during a single transient analysis to compute the RMS noise results, instead of several runs. This allows larger circuits to be handled, larger number of runs and the ability to analyze a circuit with a CPU time close to a normal (noiseless) transient analysis.

Eldo will use the MRUN algorithm (perform several runs sequentially) by forcing the **MRUN** flag instead of the default single run algorithm if at least one of the conditions below is met:

- **.PART** being used (for ADiT, OSR, and so on)
- **.CHRSIM** present
- Presence of DELAY operators, T elements, FNS elements
- Use of local truncation error algorithm (**QTRUNC** option active)
- Mixed-mode simulation (presence of **.A2D**/**.D2A** commands)
- Use of IEM algorithm
- Use of Verilog-A

Two methods exist to generate transient noise signals. One uses a sum of sine waves with random phases. It is activated when **FMIN** is not zero. The second method, only activated when **FMIN**=0, uses gaussian random variables to generate noise sources. A noise source, with given frequency characteristics, will produce two different noise signals depending on the method used to generate them. These signals will have different instantaneous values but will have the same power and same frequency content.

Note

The default values of **EPS**, **VNTOL**, **RELTOL**, and **HMAX** are changed in transient noise analysis from the values used in other analyses. For **.NOISETRAN**, the defaults are **eps**=1.0×10^{-6}, **vntol**=1×10^{-6} and **reltol**=1×10^{-6}. By default, Eldo is able to compute noise voltages down to about 1µV. As most of the applications create higher levels of noise, this should be sufficient in most cases. In all other cases, it is advisable to increase the simulator accuracy.

For **.NOISETRAN**, the default **HMAX** value is **HMAX**=1/(2×**FMAX**). It is better to explicitly set a lower value for **HMAX** than a high value for **FMAX** to achieve the best accuracy possible.

When a **.NOISETRAN** simulation is requested, transient analysis extracts (or extracts without any analysis type) will by default return only nominal values. Specify option **DEFRMSNTR** if both RMS values and nominal values should be dumped. In such a case, **.EXTRACT TRAN** extracts (or extracts without any analysis type) will return both RMS and nominal values: RMS values computed by **.NOISETRAN**, nominal values from the nominal transient analysis. This only has an effect for **.NOISETRAN** simulation with multiple run **MRUN** specified. Use the analysis type **NTR** on the **.EXTRACT** command to select computation based on RMS.

If a single-run **.NOISETRAN** is performed, then the **.EXTRACT** command without any analysis will be interpreted only for the TRAN run, not for the NOISETRAN. In this case, the **NTR** keyword on the **.EXTRACT** command must be specified. For example:

```
.NOISETRAN... MRUN
.EXTRACT <expression>
```

then expression will be evaluated for only the nominal run. If option **DEFRMSNTR** is set, the nominal and the NTR analysis (on RMS waves) will be reported. Use the keyword **NTR** on the extract to obtain the RMS extract value.

If using **.NOISETRAN** in a single run analysis, then option **INTERP** is disabled and a warning message displayed.

Use option **KEEP_HMPFILE** to generate a *.hmp* output file for transient noise analysis results. The *.hmp* file, of COU format (a legacy format), contains results for each NOISE analysis, and can be viewed with the waveform viewer. By default, it is not generated. If the **NONOM** parameter is specified, then this option is always ignored and the *.hmp* file is not generated, This is because, with **NONOM** specified, only one run, the noisy run, is performed. This is written directly to the regular *.wdb* file, without the need for an extra *.hmp* file, which is only used to compute RMS values when the simulation is complete. If not required for simulation results, the *.hmp* file should not be generated in order to save disk space.

Parameters

- **FMIN**=VAL

  Lower limit of the noise frequency band.

- **FMAX**=VAL

  Upper limit of the noise frequency band.

  FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters or oscillators and mixers that exhibit frequency conversion. **FMIN** is also used to specify the algorithm used to generate the noise sources in the time domain. When **FMIN** > 0 the noise sources are generated as a sum of **NBF** sinusoids. When **FMIN**=0 another algorithm is used, generating noise sources with a continuous spectrum between 0 and **FMAX**.

- **NBRUN**=VAL

  Defines the number of simulations which are performed with the noise sources included. This defines the accuracy of the noise analysis.

  If **NBRUN** = 1, the output voltages or currents stored in the binary output file are stored as simple voltages or currents. The noise voltages and currents of the **NBRUN** simulations are stored together in the *.wdb* output file, which may be viewed with the waveform viewer. The file contains results for each NOISE analysis.

  If **NBRUN** > 1, the binary output file will contain the RMS values of these curves. Otherwise, the binary output file will simply contain the noise voltage(s) or current(s).

  When **NONOM** is specified, **NBRUN** is ignored and a warning generated.

- **NBF**=VAL

  Specifies the number of sinusoidal sources with appropriate amplitude and frequency and with randomly distributed phase from which the noise source is composed. The default value is 50. This parameter has no effect when **FMIN** is set to 0.

- **AMP**=VAL

  This parameter is the noise source amplification factor and only affects internal noise computations. The noise is internally multiplied by this factor in order to differentiate electrical noise from numerical noise and afterwards, when RMS noise values are calculated, the values are divided by this factor again to provide correct results. This factor should only be used if the noise level is very low (for example below 1 μV). Care must be taken if this feature is used as some circuits are non-linear, even with small signals, which may cause incorrect results. The default value of **amp** is 1.

- **SEED**=VAL

  Used to initialize the random number generator. Must be an integer between 0 and $2^{31}-1$. Performing two noise simulations will provide the same RMS noise results. The default value is 0.

- **NOMOD**=0|1|2

  Switch to select the noise source for MOSFET devices:

  `0` Noise simulation with thermal and flicker noise. Default.
  `1` Noise simulation without thermal noise.
  `2` Noise simulation without flicker noise.

- **NONOM**

  No nominal simulation. Specifies that only one noisy simulation should be performed and the nominal simulation is suppressed. If specified, the *.hmp* file is never generated, even if option **KEEP_HMPFILE** is also specified. When **NONOM** is specified, **NBRUN** is ignored and a warning generated.

- **TSTART**=VAL

  Specifies the first time point from which the circuit generates noise (before `TSTART` the circuit is noiseless). Default value is 0.

- **TSTOP**=VAL

  Specifies the last time point of the transient noise analysis. After `TSTOP` the circuit no longer generates noise. Default value is `TSTOP` of the **.TRAN** command.

- **MRUN**

  Multiple run. Forces the algorithm to perform several runs sequentially to compute the RMS noise results. Specify this if you do not want the algorithm to perform parallel noise runs. Histograms will be written to the output file if the number of simulation runs is greater than the number specified by **NBBINS**.

- **ALL**

  Forces Eldo to dump the results of the simulation in the ASCII output files (by default these waves are not dumped, since they are normally unusable).

- **NBBINS**=VAL

  Specifies the number of bins for the histogram produced when the transient noise is used with the **.EXTRACT** command. Default is 10.

- **FMIN_FLICKER**=VAL

  Specifies the lower limit of the noise frequency band for the flicker noise source when **FMIN**=0. If **FMIN**>0 this parameter will be ignored. Optional.

Example

```
.NOISETRAN ...
.extract TRAN label = L1 ...
.extract      label = L2 ...
.extract  NTR label = L3 ...
```

Here only one value computed for both L1 and L2, based on the nominal run, and one value for L3 (computed from RMS values). With option **DEFRMSNTR**, two extract values will be returned for both L1 and L2, and one for L3.

Related options

See “DEFRMSNTR” on page 999, “NODEFRMSNTR” on page 1004 and “KEEP_HMPFILE” on page 1002.

.NOTRC

Suppress Netlist from an Output File

.NOTRC

This command suppresses the rewriting of the circuit description file in the ASCII output (*.chi*) file. You can specify this command anywhere in the netlist, it does not need to be placed immediately after the first line.

Example

```
example title
.notrc
r1 1 2 5
...
.end
```

Related options

See "NOTRC" on page 1005 and "NOTRCLIB" on page 1005.

.NWBLOCK

Partition Netlist into Newton Blocks

```
.NWBLOCK [RELTOL=value] [VNTOL=value] list_of_nodes
```

This command allows the user to control the way Eldo will partition the netlist into several Newton Blocks. Additionally, blocks can have different accuracies.

There can be several **.NWBLOCK** commands. Each **.NWBLOCK** will result in the creation of a Newton Block.

Parameters

- `list_of_nodes`

  The list of nodes to be assigned to a Newton Block. The wildcard character '*' is allowed as shown in the following example:

  ```
  .NWBLOCK X1.*
  ```

- **RELTOL**=value

  Controls the accuracy of the Newton Block. Optional. Same meaning as the **RELTOL** global parameter that can be specified with:
  .OPTION RELTOL=val
  The scope of visibility is limited here to the node/instances referred to in the **.NWBLOCK** command.

- **VNTOL**=value

  Controls the voltage accuracy of the simulator. Optional. Same meaning as the **VNTOL** global parameter that can be specified with:

  ```
  .OPTION VNTOL=val
  ```

  The scope of visibility is limited here to the node/instances referred to in the **.NWBLOCK** command.

Notes

- If a node is referred to in several **.NWBLOCK** commands, the node will belong to the first block created via the **.NWBLOCK** command.
- **.NWBLOCK** commands are used in transient analysis (TRAN) only. **.NWBLOCK** is ignored for DC analysis.
- For nodes which do not appear in any **.NWBLOCK** commands: if the node cannot be solved by OSR, they will be grouped into another newton block.
- Once nodes have been assigned to a Newton Block, Eldo may decide to group together some or all of the newton blocks just created, for the sake of simulation efficiency.

.OBJECTIVE

Optimization Objectives

```
.OBJECTIVE
+ EXTRACT_INFO [LABEL=NAME]
+ {$MACRO|FUNCTION}
+ OBJECTIVE_INFO
+ [SCALING_INFO]
+ [PRINT_INFO]
```

The **.OBJECTIVE** command is based on the **EXTRACT** construct, and specifically dedicated to optimization. Some restrictions are imposed and the semantic is different in some situations. The specification of design objectives has been simplified.

For the complete description of optimization capabilities, see the separate chapter Optimizer in Eldo.

.OP

DC Operating Point Calculation

```
.OP [[KEYWORD] T1 {[KEYWORD] TN}]
.OP TIME=VAL|END [STEP=VAL] [TEMP=VAL]
.OP DC=VAL [DC2=VAL] [STEP=VAL] [TEMP=VAL]
```

This command forces Eldo to determine the DC operating point of the circuit with inductors short-circuited and capacitors opened. If either the specified simulation time is reached or one of the conditions described in the “optional parameters” below is fulfilled, the operating point is saved to the *.chi* file.

If no parameter is specified, the operating point information is saved for DC prior to AC or first DC analysis in the case of a DC sweep.

Additional information concerning the operating points such as power dissipation, node voltages and source currents are written to the *.chi* output file.

The result table generated by the **.OP** command includes all OP results in addition to node voltages and device currents. These terms (`BETADC`, `BETAAC`, `CXS`, `VTH_D`) are printed out in the OP table, and can also be plotted/printed.

When performing operating point calculation at time *t*, with **.OP** `t`, Eldo saves voltages on nodes and static currents in branches, and outputs them as an Operating Point. However, since only static current is taken into account, the current results might be different from the transient current results at the same time, which also include dynamic currents (for example $i = C \times dv/dt$).

Parameters

- `KEYWORD`

  Can be one of the strings: **ALL**, **DEF**, **BRIEF**, **CURRENT**, **VOLT**.

  Select **ALL** or **DEF** (synonyms) to display OP information for both current and voltage nodes (default).

  Select **BRIEF** or **CURRENT** (synonyms) to display OP information only for currents of each element, a limited set of information compared to the **ALL**/**DEF** case.

  Select **VOLT** to display OP information only for the voltage nodes.

- `T1, TN`

  Simulation times at which operating point information will be recorded. The parameter **END** can be specified as a simulation time.

- **TIME**=VAL

  Simulation time for which **.OP** results are written.

- **END**

  Forces Eldo to output the **.OP** information after a transient analysis (**.TRAN**) if specified.

- **STEP**=VAL

  Step value for which **.OP** results are written. This is the case for when the **.STEP** command is used.

- **TEMP**=VAL

  Current temperature for which **.OP** results are written.

- **DC**=VAL

  DC sweep value for which **.OP** results are written.

- **DC2**=VAL

  Second DC value for which **.OP** results are written in cases where a double DCSWEEP analysis is performed.

Note

A **.OP** command is always automatically performed prior to an AC analysis. If no other analysis is specified, a **.OP** command forces a DC analysis to be performed after the operating point has been calculated.

For MOSFET and BJT, Eldo prints in the DCOP point table the operating region of the device. The following messages are written:

- For MOSFET:

```
if ((vgs - vgt) < 0) "SUBTHRESHOLD"
else if ((vds - vdss) < 0) "LINEAR"
else "SATURATION"
```

- For BJT:

```
if (VBC > VBCSAT)
   if (VBE > 0) "SATURATION"
   else "INVERSE"
else
   if (VBE > 0) "ON"
   else "OFF"
```

VBCSAT can be specified in the **.OPTION** command. Default is 0.

Options

- **.OPTION OPTYP**=VAL

  Used to change the way Operating Point information related to MOSFETs is displayed in the ASCII output file.

- **.OPTION OPTYP**=1

  Full operating point table is displayed. This is the default.

- **.OPTION OPTYP**=2

  Can be specified only when either the **-st** flag or the **STVER** option are set. Used to print the reduced operating point information for the MOSFET models (see the Spice documentation for more details). Otherwise equivalent to **optyp**=1. This can be used with UDM and BSIM4 models.

- **.OPTION OPTYP**=3

  Output compatible with Spice3e2.

Table 10-17 lists the character string displayed in the Operating Point table for the different **optyp** values. Elements on the same line are synonymous: that is, the same value would be printed if **optyp** changes.

Table 10-17. Operating Point—optyp values

Default	optyp=3	When -st flag or STVER option is set	When -st flag or STVER option is set & optyp=2
ID	ID	ID	ID
		IS	
		IB	
VGS	VGS	VGS	VGS
VDS	VDS	VDS	VDS
VBS	VBS	VBS	VBS
VTH	VTH	VTH	VTH
VDSAT	VDSAT	VDSAT	VDSAT
			gm
			gm
GM	GM		
GDS	GDS		
GMB	GMB		gmbs
		Ibd	Ibd
		Ibs	Ibs
		Gdd	Gbd
		Gdg	Gbs
		Gds	
		Gsd	
		Gsg	

Table 10-17. Operating Point—optyp values

Default	optyp=3	When -st flag or STVER option is set	When -st flag or STVER option is set & optyp=2
		Gss	

An explanation of some of these parameters is provided below:

Ibd Bulk-drain current through the parasitic *BD* diode

Ibs Bulk-source current through the parasitic *BS* diode

VTH Internal threshold OP dependent; see the *Eldo Device Equations Manual* for how it is computed

VSS or *VDSAT* Saturation voltage at DCOP

Gxy Derivative of static current on node *x* with respect to *Vy*; *x* and *y* can be one of *D*, *G*, *S*, *B*

Cxy Derivative of charge on node *x* with respect to *Vy*; *x* and *y* can be one of *D*, *G*, *S*, *B*

Isub Subthreshold current component

As shown in Table 10-17, with the **`-st`** flag or **`STVER`** option set and **`optyp`** not set or not 2, then Eldo reports several *Gxy* terms, where *Gxy* stands for derivative of current on pin *x* with respect to voltage on pin *y*. For example, *Gds* is the derivative of the current on the Drain with respect to the voltage on the Source. In all other cases, just three conductances values are reported:

- *GM* is the derivative of *IDS* with respect to *VGS*
- *GDS* is the derivative of *IDS* with respect to *VDS*
- *GMBS* is the derivative of *IDS* with respect to *VBS*

In the OP table for BSIM3v3, Eldo also displays *VBI* and *PHI*:

PHI Surface potential at strong inversion

VBI Built-in voltage for the PN junction between the substrate and the source

Table 10-18. Operating Point—optyp values Dynamic Part for Charge Control Model

Default	optyp=3	When -st flag or STVER option is set	When -st flag or STVER option is set & optyp=2
Cdd	Cddb	Cdd	
Cdg	Cdgb	Cdg	
Cds	Cdsb	Cds	
Cdb		Cdb	Cdb+CJDB
Cgd	Cgdb	Cgd	Cgd+CVGD
Cgg	Cggb	Cgg	
Cgs	Cgsb	Cgs	Cgs+CVGS
Cgb		Cgb	Cgb+CVGB
Csd		Csd	
Csg		Csg	
Css		Css	
Csb		Csb	Csb+CJSB
Cbd	Cdbd	Cbd	
Cbg	Cbgb	Cbg	
Cbs	Cbsb	Cbs	
Cbb		Cbb	

Table 10-19. Spice3e Capacitance / Eldo Capacitance

Spice3e capacitance	Relationship with Eldo capacitance
Cbsb	–(Cbg+Cbd+Cbb)
Cbdb	Cbd
Cbgb	Cbg
Cdsb	(Cgg+Cgd+Cgb+Cbg+Cbd+Cbb+Csg+Csd+Csb)
Cddb	–(Cgd+Cbd+Csd)
Cdgb	–(Cgg+Cbg+Csg)
Cgsb	–(Cgg+Cgd+Cgb)

Table 10-19. Spice3e Capacitance / Eldo Capacitance

Spice3e capacitance	Relationship with Eldo capacitance
Cgdb	Cgd
Cggb	Cgg

In SPICE operating point display the ***Cbs*** and ***Cbd*** values are bulk-source and bulk-drain currents correspondingly. These results do not correspond to Eldo ***Cbs*** and ***Cbd*** since these last two items correspond to capacitances.

Example

```
R1 N1 N2 R1
.PARAM R1=1k
.STEP PARAM R1 1k 10k 1k
.TRAN 1n 100n
.OP
.OP TIME=9n STEP=5k
.END
```

In the above example, an Operating Point will be determined for a resistance of 5kΩ at time 9ns.

Note

The **`.op`** command may be used to run AC and NOISE analyses from within a **`.tran`** command at specified times, see "AC in the middle of a .TRAN" on page 543 for more details.

.OP_DISPLAY

DC Operating Point Display

```
.OP_DISPLAY
+ [MODEL=model_name]
+ [VTMOD=0|1|2]
+ [VTVDS=vds]
+ [VTCIDS=ids_norm]
+ [VTVGSMIN=vgsmin]
+ [VTVGSMAX=vgsmax]
```

This command enables the printing of alternative threshold voltage (VT) values for MOS devices. It is only active when a .OP command is specified. The VT results are written to the OP section of the ASCII *.chi* output file. The alternative calculation methods are: maximum transconductance algorithm threshold (MAXGMVT) and constant current threshold (VTC).

These alternative methods ensure the electrical test limits of MOSFETs in manufacturing are aligned with the SPICE models. By default, the standard algorithm is based only on model parameters and is not directly measurable on a physical transistor.

If no parameters are specified, or if **VTMOD** is set to 0 (see below), the standard method of calculating VT is used, so the command is ignored. Multiple commands can be specified to display different reports, that is, it is possible to print both MAXGMVT and VTC in the *.chi* file.

Parameters

- **MODEL**=model_name

  Specifies the models to which this command applies. `model_name` can contain the wildcard character '*'. If this keyword is not specified, the command will be applied to all models, equivalent to specifying **MODEL**=*.

- **VTMOD**=0|1|2

  Specifies the alternative method to calculate VT.

  VTMOD=0
  Eldo standard algorithm. This is the default value.

  VTMOD=1
  Eldo will calculate and report MAXGMVT.

  VTMOD=2
  Eldo will calculate and report VTC.

- **VTVDS**

  Specifies the default VDS (drain source voltage) value for Eldo to use in the VT calculations. See the MAXGMVT, VDSATC and VTC extract functions for further details. Default value is 0.1.

- **VTCIDS**

  Specifies the default IDS_NORM (normalized drain source current) value for Eldo to use in the VT calculations. See the VDSATC and VTC extract functions for further details. Default value is 600 nA.

- **VTVGSMIN**

  Specifies the default VGSMIN (minimum gate source voltage) value for Eldo to use in the VT calculations. See the MAXGMVT, VDSATC and VTC extract functions for further details. Default value is LV9(Mxx) – 0.5.

- **VTVGSMAX**

  Specifies the default VGSMAX (maximum gate source voltage) value for Eldo to use in the VT calculations. See the MAXGMVT, VDSATC and VTC extract functions for further details. Default value is LV9(Mxx) + 0.5.

Example

```
.op

.op_display model=*
+ VTMOD=1
+ VTVDS=0.1
+ VTCIDS=600e-9

.op_display model=*
+ VTMOD=2
```

Example extract of *.chi* file:

```
          X1.M1           X2.M1           X3.M1           X4.M1

MODEL     NSHORT.3        NSHORT.3        NSHORT.3        NSHORT.3
ID         0.0             0.0             0.0             4.56889437E-08
Ibd        0.0             0.0             0.0            -1.00000000E-14
Ibs        0.0             0.0             0.0             0.0
VGS        3.00000000E+00  0.0             0.0             0.0
VDS        0.0             0.0             0.0             3.00000000E+00
VBS        0.0             0.0             0.0             0.0
VTH        3.67156515E-01  3.67156515E-01  3.67156515E-01  3.63441805E-01
VDSAT      1.54552476E+00  4.53584187E-02  4.53584187E-02  4.53582818E-02
[...]
Region    linear          subthreshold    subthreshold    subthreshold
VTH_D      2.63284349E+00 -3.67156515E-01 -3.67156515E-01 -3.63441805E-01
VTC        3.67032691E-01  3.67032691E-01  3.67032691E-01  3.67032691E-01
MAXGMVT    3.28272746E-01  3.28272746E-01  3.28272746E-01  3.28272746E-01

          X1.M2           X2.M2           X3.M2           X4.M2

MODEL     PSHORT.1        PSHORT.1        PSHORT.1        PSHORT.1
ID         0.0             0.0             0.0             5.94230571E-03
Ibd        0.0             0.0             0.0            -6.87153080E+00
Ibs        0.0             0.0             0.0             0.0
VGS        3.00000000E+00  0.0             0.0             0.0
VDS        0.0             0.0             0.0             3.00000000E+00
```

```
VBS        0.0             0.0             0.0             0.0
VTH       -3.57021910E-01 -3.57021910E-01 -3.57021910E-01 -3.20172698E-01
VDSAT     -4.39505891E-02 -4.39532068E-02 -4.39532068E-02 -2.04057023E+00
[...]
Region     subthreshold    subthreshold    subthreshold    subthreshold
VTH_D     -3.35702191E+00 -3.57021910E-01 -3.57021910E-01 -3.20172698E-01
VTC       -3.53187113E-01 -3.53187113E-01 -3.53187113E-01 -3.53187113E-01
MAXGMVT   -3.29728569E-01 -3.29728569E-01 -3.29728569E-01 -3.29728569E-01
```

.OPTFOUR

FFT Post-processor Options

```
.OPTFOUR [TSTART=VAL|EXPR] [TSTOP=VAL|EXPR] [NBPT=VAL] [FS=VAL]
+ [NORMALIZED=0|1] [NORMALIZATION_FACTOR=val] [INTERPOLATE=0|1|2|3]
+ [NOROUNDING[=1]] [RAPWIN=VAL]
+ [WINDOW=name] [ALPHA=VAL] [BETA=VAL] [PADDING=1|2|3]
+ [FMIN=VAL] [FMAX=VAL] [FNORMAL=freq] [DISPLAY_INPUT=0|1]
```

Used to supply the options for the FFT post-processor. See also ".FOUR" on page 673.

Setting Parameters

The following four parameters are used to specify the time window on which the FFT will be performed. This can be summarized by Table 10-20, which provides the values of TSTART, TSTOP, FS, NBPT for all possible cases.

- **TSTART**

  Time value from which the time window will start. Default value is option STARTSMP if specified, or TSTART from .TRAN, or TSTOP-NBPT/FS if all three other parameters specified.

- **TSTOP**

  Time value that specifies the end of the time window. Default value is TSTART from .TRAN, or TSTART+NBPT/FS if all three other parameters specified.

- **FS**

  Sampling frequency. Default value is NBPT/(TSTOP-TSTART).

- **NBPT**

  Number of points to be used in the time window. Default value is 1024, or (TSTOP-TSTART)×FS if FS specified.

Table 10-20. Default Values for Time Window Parameters

TSTART	TSTOP	FS	NBPT
User Defined	TSTOP	**NBPT**/(**TSTOP-TSTART**)	1024
STARTSMP or **TSTART**	User Defined	**NBPT**/(**TSTOP-TSTART**)	1024
STARTSMP or **TSTART**	TSTOP	User Defined	(TSTOP-TSTART)*FS
STARTSMP or **TSTART**	TSTOP	**NBPT**/(**TSTOP-TSTART**)	User Defined
User Defined	User Defined	**NBPT**/(**TSTOP-TSTART**)	1024
User Defined	TSTOP	User Defined	(TSTOP-TSTART)*FS
User Defined	TSTOP	**NBPT**/(**TSTOP-TSTART**)	User Defined

Table 10-20. Default Values for Time Window Parameters

TSTART	TSTOP	FS	NBPT
STARTSMP or **TSTART**	User Defined	User Defined	(TSTOP-TSTART)*FS
STARTSMP or **TSTART**	User Defined	**NBPT**/(**TSTOP-TSTART**)	User Defined
STARTSMP or **TSTART**	TSTART+NBPT /FS	User Defined	User Defined
User Defined	User Defined	User Defined	(TSTOP-TSTART)*FS
User Defined	User Defined	**NBPT**/(**TSTOP-TSTART**)	User Defined
User Defined	TSTART+NBPT /FS	User Defined	User Defined
TSTOP-NBPT/FS	User Defined	User Defined	User Defined
User Defined	User Defined	User Defined	User Defined

Note

If both **FS** and **NBPT** parameters are specified by the user then **NBPT** will be calculated using the relation **NBPT** = (**TSTOP** - **TSTART**) * **FS**.

Note

PADDING parameter can be specified if **TSTART**, **TSTOP**, **FS** and **NBPT** are used *and* **NBPT** > (**TSTOP**-**TSTART**)*__FS__.

- **NORMALIZED**

  Specifying 1 means that all the points of the result are multiplied by 2/**NBPT**. Default. Specifying 0 means no normalization of the results.

- **NORMALIZATION_FACTOR**

  Replaces the default factor (2/**NBPT**) which is applied to FFT results when **NORMALIZED** is set to 1. It is ignored if **NORMALIZED** is set to 0.

- **INTERPOLATE**

  Sets type of interpolation:

  0 = No interpolation (default)
  1 = Linear interpolation
  2 = Cubic Spline interpolation
  3 = Blocker Sampler interpolation

- **NOROUNDING[**=1**]**

  By default, Eldo truncates and rounds **TSTART** and **TSTOP** values at 1ps. If keyword **NOROUNDING** is specified, this rounding is not performed.

- **RAPWIN**

  Represents a fraction of the FFT window. See description at end of Notes section below.

Notes

1. FFT is much faster when the **NBPT** parameter (either user given, or computed from the three other parameters) can be a product of powers of 2, 3 and/or 5 so that fast algorithms can apply. For example, 1022 can be divided by 2, but is not a good candidate; $1024=2^{10}$ or $960=3\times5\times2^{6}$ are better candidates. In such cases, the FFT program will make use of several optimizations which do not alter the FFT results.

2. The last time point value actually taken into account by FFT is the value at time: **TSTOP**–1.0/**FS**. This is to deal with the fact that when FFT is done on a circuit which is in steady-state mode, time value F(**TSTOP**) equals the time value F(**TSTART**), and hence that last value must not be used for the FFT since FFT must be computed on a full number of periods.

3. **TSTART** and **TSTOP** time values can be specified as results of an expression that follows the **.EXTRACT** syntax, see ".EXTRACT" on page 637. When the expression can be measured, FFT sampling will start.

Caution

When the parameter **TSTOP** is specified as an expression and the parameter **NBPT** is used, then it is impossible for Eldo to determine the sampling frequency before the simulation. Furthermore, Eldo cannot compute exactly the points which will be used to compute the FFT. Therefore, the parameter **INTERPOLATE** cannot be set to 0 (no interpolation). In this case a warning message will be displayed by Eldo and the parameter **INTERPOLATE** will be set to 2.

4. A very important point is related to the **INTERPOLATE** parameter: FFT must be done on equally-spaced time value points. However, the time points computed by an analog simulator such as Eldo are not equally spaced. There are two possibilities to obtain the equally-spaced points required by FFT:

   - Force Eldo to compute values at least on the points requested by the FFT (and in between Eldo can do smaller time steps if needed),

   - Interpolate between the time values points computed 'freely' by Eldo: this interpolation can be a LINEAR interpolation (**INTERPOLATE** = 1) or a higher order interpolation (**INTERPOLATE** = 2: a cubic SPLINE is used).

FFT results are better when no interpolation occurs. For this reason the parameter **INTERPOLATE** defaults to 0. In such a case, Eldo will compute time-value points at least every 1/**FS** seconds.

If **INTERPOLATE** is set to 1 or 2, Eldo will compute its time steps according to the activity in the design and the equally-spaced time points required by the FFT will be interpolated from that time-domain waveform generated by Eldo. **INTERPOLATE**=2 gives better results than **INTERPOLATE**=1.

The **.OPTFOUR INTERPOLATE** command and **.OPTION FREQSMP** are independent (prior to the 2008.1 release they were not). With this scheme, and with **FREQSMP** not set, then FFT results might be not as accurate as before, because the time step is only computed very accurately inside the FFT window. The FFT results might be not accurate if the signal has not been computed with enough accuracy when arriving at the beginning of the window. Use parameter **RAPWIN** to handle this, which represents a fraction of the FFT window: if positive, Eldo will start applying the time constraint which will prevail in the FFT window [**TSTART**, **TSTOP**] in the window [**TSTART** − **RAPWIN** × (**TSTOP** − **TSTART**), **TSTART**]. Alternatively, it is possible to use **FREQSMP**, or **HMAX**, to impose some constraints to the time step outside the FFT window.

PADDING Parameter

Specifies where to add zeros inside the FFT input window: only used if **TSTART**, **TSTOP**, **NBPT** and **FS** are given and **NBPT** > (**TSTOP-TSTART**) × **FS**.

PADDING can be set to:

1	Zeros are added at the beginning of the FFT input window
2	Zeros are added at the end of the FFT input window
3	Zeros are added evenly at the beginning and at the end of the FFT input window.

Display Parameters

- **FMIN**

  Starting frequency used inside the FFT result window.

- **FMAX**

  Last frequency used inside the FFT result window.

Note

FMIN and **FMAX** parameters are not used for an FFT on an FMODSST signal.

- **FNORMAL**=freq

  Adjusts the results around the Y-axis so that the point for the specified frequency is 0.0.

- **DISPLAY_INPUT**

  Generates an additional transient waveform of the exact data points used for the FFT calculation. This FFT_INPUT waveform contains equally-spaced time value points from START time to one datapoint short of the STOP time.

  0 = do not create the extra FFT_INPUT transient waveform (default)

1 = create the FFT_INPUT waveform if a **.PLOT** or **.PROBE** command exists for the **.FOUR** item. For example:

```
.four label=test v(foo)
.plot four test
```

with **DISPLAY_INPUT**=1 the FFT_INPUT(TEST) waveform will be generated inside the TRAN folder in the EZwave Waveform List panel.

See Performing a Fast Fourier Transform in the *EZwave User's and Reference Manual*.

Sampling WINDOW Parameter

Specifies the Sampling Window to be used. The following parameters are allowed:

RECTANGULAR (default)
PARZEN
BARTLETT
WELCH
BLACKMAN
BLACKMAN7
KLEIN
HAMMING
HANNING
KAISER
DOLPH_CHEBYCHEV

Associated Parameters

- **ALPHA**

  Used when the **WINDOW** is **DOLPH_CHEBYCHEV** or **HANNING**.

- **BETA**

  Used when the **WINDOW** is **KAISER**. Constant which specifies a frequency trade-off between the peak height of the side lobe ripples and the width of energy in the main lobe.

Please refer to the *EZwave User's Manual* for further details of the windowing equations.

Example

```
.optfour tstart=xdown(V(1),2u,1) tstop=xdown(V(1),2u,1000) ..
.four ...
```

Eldo computes an FFT with 1000 periods of signal `V(1)`. This shows how **TSTART** and **TSTOP** time values can be specified as results of an expression. When the expression can be measured, FFT sampling will start.

```
.optfour tstart=0 tstop={xdown(V(1),10m,13)} nbpt=10000
```

When the parameter `tstop` is specified as an expression and the parameter `nbpt` is used, then it is impossible for Eldo to determine the sampling frequency before the simulation. Eldo cannot compute exactly the points which will be used to compute the FFT. Therefore, the parameter **INTERPOLATE** cannot be set to 0 (no interpolation). In this case the following warning message will be displayed by Eldo and the parameter **INTERPOLATE** will be set to 2.

```
Warning: .OPTFOUR : parameter INTERPOLATE is set to 2 because NBPT is
given and TSTOP is an expression
```

.OPTIMIZE

Optimization

```
.OPTIMIZE [qualifier=value {, qualifier=value }]
+          [PARAM=list_of_parameters | *]
+          [RESULTS=list_of_targets | *]
```

The global specification of an optimization configuration acting on all the analyses specified in the circuit netlist is done using the **.OPTIMIZE** command.

For the complete description of optimization capabilities, see the separate chapter Optimizer in Eldo.

.OPTION

Simulator Configuration

```
.OPT[ION] OPTION[=VAL] {OPTION[=VAL]}
```

The **.OPTION** command allows the user to modify Eldo execution behavior by allowing the setting of parameter values other than the default ones.

For the complete description of options available, see the separate chapter "Simulator and Control Options" on page 939.

.OPTNOISE

AC Noise Analysis

```
.OPTNOISE [ALL ON|OFF] [<CLASS> ON|OFF]
+ [R ON|OFF|<max>] [OUTSOURCE ON|OFF] [NSWEIGHT <FILENAME>]
+ [SORT D|V|TD|TV [SN <n>|SV <value>]] [NBW <FMIN> <FMAX>]
```

.OPTNOISE allows more flexibility in the output of the AC noise analysis results: noisy elements can be sorted in different ways, and a weight function can be applied before printing out the noise contribution. The **.OPTNOISE** command must be used in conjunction with the **.AC** and **.NOISE** analyses. **.OPTNOISE** has no effect on **.SSTNOISE** analysis and results.

Parameters

- **ALL**

  Specifies that all devices should contribute to the total noise.

- CLASS

  Is one of the following keywords: **MOS**, **BJT**, **NPN**, **PNP**, **NMOS**, **PMOS**, **DIODE**, **JFET**, **NJF**, **PJF**. For example: MOS OFF means all MOS devices source noise is turned OFF.

- **OUTSOURCE**

  Specifies that printing out of NOISE information in the ASCII output file must be turned ON or OFF.

- **NSWEIGHT** <filename>

  Reads a file for weighted functions: the format of this file is shown in "Format of files containing weight" on page 773.

- **R**

  ON: all resistors are assumed to be noisy.

  OFF: all resistors are assumed to be noiseless

  MAX: all resistors above MAX are assumed to be noisy.

- **SORT**

  Choose the way the information is displayed in the ASCII output file:

  D: sort by device,

  V: sort by value,

  TD: sort by technology (CLASS), and in each CLASS by device,

  TV: sort by technology (CLASS), and in each CLASS by value,

  SN <n>: list the highest n contributions: default n=20

  SV <value>: list the device contribution until value in % of the total noise is exceeded. Default value=0.95 (95%).

- **NBW**

  stands for Noise BandWidth, which can be different than that of AC band: RMS Average is computed on this bandwidth which defaults to the AC bandwidth. FMIN and FMAX values define the frequency band of this noise bandwidth.

Multiple output noise

Several **.NOISE** commands can be given for the same run. Eldo will then perform as many NOISE analyses as required. See the OUTPUT/post-processing section below for reporting information.

Output/Post-processing

- A noise table that contains general information is first printed out: for each output node there appears the noise RMS value, the Average and total noise power, the maximum and minimum noise power contribution, and the frequency at which these last two values are obtained. In case the weighting function is applied, both weighted and unweighted values are dumped, but for other information below, just weighted or unweighted values will come depending on the content of the **.OPTNOISE** command.
- For each output noise, total noise source values are displayed, sorted according to the **.OPTNOISE** specification (**SORT SN** or **SORT SV** arguments).
- For each output and each listed element which appears in 2), the table total noise=f(FREQ) is printed out, each <n> frequency points, where n is the number specified in the **.NOISE** command. Here, frequencies are those of AC: an asterisk (*) should appear to mark the frequency corresponding to Noise Bandwidth. If there is no sorting of information, then the regular SPICE output will be printed out, with detail of the contribution of each device.
- Regular frequency output table: this is the content of **.PRINT NOISE**.
- A file *<namecir>_nsa.cou* will also be created, containing the noise response of the devices selected by the **SORT** command option.
- By default, the noise response of devices selected by the **SORT** command option are merged into the NOISE folder of the *.wdb* file. Use option **NSAFILE_FORMAT** to control the output generated. Set to **COU** for backward compatiblity (pre-2009.2) behavior to create a separate *<netlist>_nsa.cou* file. Set to **NONE** to disable the creation of this file if you are only interested in the table inside the *.chi* file. Default is **WDB**.

Format of files containing weight

References:

1. IEEE Standard Methods and Equipment for Measuring the Transmission Characteristics of Analog Voice Frequency Circuits. (IEEE Std. 743-1984)
2. SCAMPER Reference Manual, Rel.501, 1990 (noise analysis and option)
3. SCAMPER User’s Guide, Rel.501,1990

C-Message filter is a frequency-weighting characteristic, used for measurement of noise in voice frequency communications circuits and designed to weight noise frequencies in proportion to their perceived annoyance effect to a typical listener in telephone service. Reference point 1000 Hz.

Format of the noise weight table:

```
FreqS|* FreqE|* <flatweight> <weightunit>
+ <freq_interpolation_type>
f1 w1
...
fn wn
```

- `FreqS`

  Starting frequency for noise weight. If "*" is given then the first frequency in the freq/weight data table will be taken as the starting frequency. Units in Hertz.

- `FreqE`

  Ending frequency for noise weight. If "*" is given then the last frequency in the freq/weight data table will be taken as the ending frequency. Units in Hertz.

- `flatweight`

  If the flatweight value is given, then it will be taken as a constant (flat) noise weight over the given frequency band (no noise outside the band). The freq/weight data table, if any, will be ignored. If "*" is given in place of `FreqS` and/or `FreqE`, then the corresponding value will be taken from the `$frequency` line in the scamper netlist.

- `weightunit`

  Units of the flatweight or the weight values in the freq/weight data table:

  `DBL` weight given in decibels loss (default)

  `DB` weight given in decibels

  `MAG` weight given in numerical value (for noise power)

- `freq_interpolation_type`

  Type of interpolation for noise weight between frequencies.

  `LOG` logarithmic frequency interpolation (default)

  `LIN` linear frequency interpolation

- `fi wi`

  Table of frequency and associated weighting values. More than one pair could be given on one line.

Example weight file

```
----------------------
' this is a comment
100   4000    DBL    LOG  ' weight is in DBL, LOG interpolation
```

```
60     55.7      '   2
100    42.5      '   2
200    25.1      '   2
300    16.3      '   2
400    11.2      '   1
500    7.7       '   1
600    5.0       '   1
700    2.8       '   1
800    1.3       '   1
900    0.3       '   1
1000   0.0       '   0     ' reference point
1200   0.4       '   1
1300   0.7       '   1
1500   1.2       '   1
1800   1.3       '   1
2000   1.1       '   1
2500   1.1       '   1
2800   2.0       '   1
3000   3.0       '   2
3300   5.1       '   2
3500   7.1       '   2
4000   14.6      '   3
4500   22.3      '   3
5000   28.7      '   3
```

Table noise input source

It is possible to provide Eldo some input noise source values depending on the frequency via the **TABLE** keyword. Eldo accepts the following:

```
Ixx P1 P2 NOISE TABLE [DEC|LOG|LIN] (f1,val1) (f2,val2)...
Vxx P1 P2 NOISE TABLE [DEC|LOG|LIN] (f1,val1) (f2,val2)...
```

Eldo will assume zero current source or zero voltage source in any mode other than NOISE (that is, for DC, AC, TRAN, and so on).

For further details, please refer to the “Independent Voltage Source” on page 310 and also the “Independent Current Source” on page 317.

.OPTPWL

Accuracy by Time Window

```
.OPTPWL PARAM=((TIME1,VALUE1) (TIME2,VALUE2)...)
+ PARAM=((TIME1,VALUE1)...))
```

The **.OPTPWL** command allows some precision parameters to be reset during transient simulation for different time windows. `<PARAM>` can be any one of a number of parameters. Values are given in a piecewise-linear format. **.OPTPWL** can also be changed in Eldo interactive mode.

Parameters

- PARAM

  This can be any of the following parameters: **EPS**, **RELTOL**, **RELTRUNC**, **VNTOL**, **CHGTOL**, **HMAX**, **HMIN**, **ABSTOL**, **FLUXTOL**, **NGTOL**, **OUT_RESOL**, **ITOL**, **FREQFFT**, **PCS**, **TUNING**.

The **FREQFFT** value can be applied on the time-domain window only if **.OPTFOUR** is not active and if there is no digital simulator connected.

In the **.OPTPWL** command, FREQFFT must be used in place of the keyword FREQSMP which is used in the **.OPTION** command, that is, FREQFFT in **.OPTPWL** is equivalent to FREQSMP in **.OPTION**.

The **PCS** option can be used in conjunction with the **.OPTPWL** command to specify the time after which to turn on PCS. This might be useful when the PCS option is used in the case of non-periodic conditions which may even slowdown the simulation a little.

Example

```
.OPTPWL RELTOL=(0,1.0e-4) (10n,1.0e-3)
.OPTION RELTOL=1.0e-5
```

In the above example, during DC analysis, the value 1.0e-5 will be used, and during Transient analysis a value of 1.0e-4 will be used until 10n, 1.0e-3 will be used thereafter.

Note

TIME1 can be 0, in such case VALUE is used for DC, and would overwrite the default value or the value given via **.OPTION**. TIMEx,VALUEx can be parameters set via **.PARAM** commands. **.OPTPWL** only works in TRAN.

Please also refer to ".OPTWIND" on page 777.

.OPTWIND

Accuracy by Time Window

```
.OPTWIND PARAM=(TIME1,TIME2,VALUE1)... (TIME1N,TIME2N,VALUEN)
```

The **.OPTWIND** command allows some precision parameters to be reset during transient simulation for different time windows. `<PARAM>` can be any one of a number of parameters. Values are given per window time. **.OPTWIND** can also be changed in Eldo interactive mode.

Parameters

- PARAM

  This can be any of the following parameters: **EPS**, **RELTOL**, **RELTRUNC**, **VNTOL**, **CHGTOL**, **HMAX**, **HMIN**, **ABSTOL**, **FLUXTOL**, **NGTOL**, **OUT_RESOL**, **ITOL**, **FREQFFT**, **PCS**, **TUNING**.

The FREQFFT value can be applied on the time-domain window only if **.OPTFOUR** is not active and if there is no digital simulator connected.

In the **.OPTWIND** command, FREQFFT must be used in place of the keyword FREQSMP which is used in the **.OPTION** command, that is, FREQFFT in **.OPTWIND** is equivalent to FREQSMP in **.OPTION**.

The **PCS** option can be used in conjunction with the **.OPTWIND** command to specify the time after which to turn on PCS. This might be useful when the PCS option is used in the case of non-periodic conditions which may even slowdown the simulation a little.

Example

```
.OPTWIND RELTOL=(0,10n,1.0e-4) (100n,200n,1.0e-3)
.OPTION RELTOL=1.0e-5
```

In this example, during DC analysis, the value 1.0e-4 will be used, and during Transient analysis a value of 1.0e-4 will be used until 10n, then the default value will be used until 100n (that is, 1.0e-5), 1.0e-3 will be used between 100n and 200n, and 1.0e-5 again after 200n.

Note

TIME1 can be 0, in such case VALUE is used for DC, and would overwrite the default value or the value given via **.OPTION**. TIMEx, VALUEx can be parameters set via **.PARAM** commands. For **.OPTWIND** command only, the very last specification can be: (<TIME>,,<VALUE>) which means that after time <TIME>, the parameter value will be equal to <VALUE>. **.OPTWIND** only works in TRAN.

Please also refer to ".OPTPWL" on page 776.

.PARAM

Global Declarations

Simple Description

```
.PARAM PAR=VAL {PAR=VAL}
```

Algebraic Description

```
.PARAM PAR=EXPR {PAR=EXPR}
```

Assigning a Character String

```
.PARAM PAR="NAME"
```

User Defined Function

```
.PARAM PAR(a,b)=EXPR
```

Monte Carlo Analysis Parameters

```
.PARAM PAR=VAL|PAR=EXPR
+ LOT|DEV[/GAUSS|/UNIFORM|/USERDIST]=VAL|(dtype,-3sig,+3sig
+ [,bi,-dz,+dz [,off,sv] [,scale])
.PARAM PAR=VAL LOTGROUP=my_lot_group
.PARAM PAR=MC_DISTRIBUTION
.PARAM PAR=VAL DEVX=VAL
```

.PARAM is used to assign values to parameter variables used in model and device instantiation statements. Parameters and expressions may be used in all of the following cases, for definition and value updating of:

- Device and Model Values.
- Independent Voltage and Current Source Values.
- Linear, Polynomial Coefficients, and Arithmetic Expressions (E & G only) in Dependent Sources.
- Terms used in **.DEFMAC**, **.DEFWAVE** and **.EXTRACT** statements.
- Monte Carlo analysis distribution.

Multiple parameters can be set in a single **.PARAM** command and any combination of parameter types above may be used. There is no limit to the number of parameters that can be set.

.DC and **.STEP** statements may be used to define any parameter in the main circuit.

Note

Parameters and expressions are not allowed in device names and nodes. In commands they are only allowed if explicitly specified in the documentation. Only one definition per parameter is allowed.

If a parameter is specified more than once in a netlist, Eldo will by default use the last value specified. This includes instances that are specified earlier than the last **.PARAM** specification. Specify option **USEFIRSTDEF** to force Eldo to only use the first definition (any further definitions are ignored). For example, if a netlist includes:

```
.param res=10
r1 4 2 'res'
.param res=13
```

then r1 will take the value 13. With option **USEFIRSTDEF** specified, Eldo will only use the first definition, so r1 will take the value 10.

The **.PARAM** command is order dependent in that all components of any arithmetic expressions used must have been defined earlier in the netlist.

Note

If a parameter **P** is referred to in a netlist but not defined, Eldo searches for **P** in the **.LIB** files. Only global **.PARAM** definitions are considered. Parameter declarations within a **.SUBCKT** definition will not be considered outside that subcircuit. The **LOT** and **DEV** specifications affect only the parameter before them.

Expressions can be used in a netlist with certain restrictions. These expressions must be contained within braces **{ }**. Constants and parameters may be used in expressions, together with the built-in functions and operators.

Table 10-21 lists reserved keywords which must never appear in a **.PARAM** command. For example, the following statement generates an error:

```
.PARAM SCALE=VAL
```

Table 10-21. Reserved Keywords Never Available in .PARAM

DCM	**FREQ**	**SCALE**	**TEMP**	**TIME**
TEMPER	**XAXIS**			

Table 10-22 lists reserved keywords which cannot be specified in a **.PARAM** command if an RF analysis is specified in the netlist. If an RF analysis is specified in the netlist, and if any **.PARAM** command includes one of these reserved keywords, the command will be rejected and an error message given.

Table 10-22. Reserved Keywords Not Available in .PARAM if an RF Analysis is Specified in the Netlist

BFACTOR	**BOPT**	**B_OPT**	**FUND_OSC**	**FUND_OSC**<n>[1]
GA	**GA_mag**	**GA_dB**	**GAC**	**GAM**
GAM_mag	**GAM_dB**	**GAMMA_OPT**	**GAMMA_OPT_MAG**	**GASM**

Table 10-22. Reserved Keywords Not Available in .PARAM if an RF Analysis is Specified in the Netlist

GASM_mag	GASM_dB	GAUM	GAUM_mag	GAUM_dB
GOPT	GP	GP_mag	GP_dB	GPC
HERTZ[2]	INOISE	KFACTOR	LSC	LT_JITTER
MUFACTOR	MUFACTOR_L	MUFACTOR_S	NFMIN	NFMIN_mag
NFMIN_dB	ONOISE	PHI_OPT	RNEQ	SNF
SNF_mag	SNF_dB	SSC	SSTSNF	TGP
TGP_mag	TGP_dB	TNOM[3]	YOPT	Y_OPT

1. Where <n> is a number.
2. **HERTZ** is only unavailable when running in -compat mode.
3. **TNOM** may be specified as a parameter in a **.PARAM** command when **.OPTION DEFPTNOM** is set. The temperature value used by the Eldo model evaluator is always that which is set with **.OPTION TNOM=**VAL. **TNOM** can also be specified when running in -compat mode.

Parameters

- PAR=VAL

  Name and value of the parameter.

- PAR=EXPR

  Name of the parameter and regular arithmetic expression describing it. This expression may use parameter names already defined in the netlist, unless **LOT** or **DEV** are specified. Waves must not be part of the expression as the expression is evaluated prior to the simulation. The parameters can also depend on V or I.

For a list of the available arithmetic functions refer to "Arithmetic Functions" on page 78.

- PAR="NAME"

  Assigns a character string to the parameter. May be used to parametrize models and subcircuits. Eldo accepts quoted character strings as parameter values. These string values may be used for model names and filenames. To use a string as a parameter, enclose the string within double quotes. To maintain the case of the string enclose the string within double quotes first and then enclose within single quotes.

  The value of the string is retrieved simply by specifying the dollar sign ($) and parentheses ( ).

  See "String Parameters" on page 71 for more information.

- PAR(a,b)=EXPR

  Specifies a user-defined function in order to define a parameter using an expression. The parameter may then be called when required, for example:

```
.param P(a,b)=expression
R1 1 2 'P(a,b)'
```

- PAR=VAL|PAR=EXPR
  + **LOT**|**DEV**[**/GAUSS**|**/UNIFORM**|**/USERDIST**]=VAL|
  + (dtype,-3sig,+3sig
  + [,**bi**,-dz,+dz [,**off**,sv] [,scale]])

  This parameter setting can only be used with a Monte Carlo analysis. For more information see ".MC" on page 706.

  LOT|**DEV**=VAL

  Specifies a **LOT** or **DEV** tolerance value of VAL to the parameter for MC analysis. **LOT** causes devices to vary with each other. **DEV** causes devices to vary independently of each other. May be used in combination with **GAUSS**, **UNIFORM** or **USERDIST** which specify Gaussian, uniform or user-defined distributions respectively. VAL may be specified as a percentage using the % sign, or as an absolute value. Parametric expressions are allowed in the value of VAL, but not in the parameter defined after a **.PARAM** statement. **LOT** and **DEV** do not allow other parameters to be specified within parameters.

  For more information on **LOT** and **DEV**, see "Tolerance Setting Using DEV, DEVX or LOT" on page 711.

  LOT/GAUSS=VAL|**DEV/GAUSS**=VAL

  Specifies a Gaussian distribution of random numbers between 4σ. VAL specifies the standard deviation of the distribution around the nominal value.

  LOT/UNIFORM=VAL|**DEV/UNIFORM**=VAL

  Specifies a Uniform distribution of random numbers between σ. VAL specifies the standard deviation of the distribution around the nominal value.

  LOT/USERDIST=VAL|**DEV/USERDIST**=VAL

  Specifies a user-defined distribution of random numbers between σ. VAL specifies the standard deviation of the distribution around the nominal value.

  See "Assign Nominal Parameter Values" on page 1211.

  Different entities are able to share the same distribution. Anywhere Eldo accepts **LOT**/**DEV** specifications, you can specify **LOTGROUP**=group_name.

  If no distribution type is specified it will default to **UNIFORM**. For more information on user defined distributions see ".DISTRIB" on page 614.

  dtype
  nor for gaussian distribution.
  uni for uniform distribution.

  -3sig
  Lower 3 sigma bounds with respect to nominal value, this can be specified as a percentage or an absolute value.

`+3sig`
: Upper 3 sigma bounds with respect to nominal value, this can be specified as a percentage or an absolute value.

`bi`
: An optional pair of characters specifying that the distribution is bimodal.

`-dz`
: Lower limit of the "dead zone" in bimodal distribution, this can be specified as a percentage or an absolute value.

`+dz`
: Upper limit of the "dead zone" in bimodal distribution, this can be specified as a percentage or an absolute value.

`off`
: An optional offset.

`sv`
: A percentage or absolute value that moves the nominal value of a parameter either above or below the "typical" nominal value for that parameter.

`scale`
: Specifies whether the calculations are to be held in log or linear scale. The two options are **`lin`** or **`log`**.

Note

1. The limits for this syntax for Accusim are `-3sig`, `3sig` compared to those of Eldo which are `-4sig`, `4sig`.
2. The minus sign in the values `-3sig` and `-dz` is only to specify that they are to the left of the nominal value. Eldo also accepts them as positive values.
3. `sv` can be > 0 which means a shift to the right, or < 0 which means a shift to the left.

- **`LOTGROUP`**`=my_lot_group`

  Different entities are able to share the same distribution. Anywhere Eldo accepts **`LOT`**/**`DEV`** specifications, you can specify **`LOTGROUP`**`=my_lot_group`. Please refer to ".LOTGROUP" on page 701 for more information.

- `PAR=MC_DISTRIBUTION`

  When using a Monte Carlo analysis the same random variable will be used each time a parameter affects a model parameter (**`LOT`** variation). When a declared parameter affects an instance parameter a new random variable is calculated each time it is specified (**`DEV`** variation).

Note

It is possible to specify that **`DEV`** variation will be used for both model and instance parameters, as was default in Eldo versions prior to v6.3_2, by specifying option **`PODEV`**. See "PODEV" on page 982 for more information.

A Monte Carlo distribution can be used to specify how the random variables should be distributed between its upper and lower limits. For more information on Monte Carlo analysis, see "Monte Carlo Analysis" on page 1207.

`MC_DISTRIBUTION` should be replaced with one of the following keyword statements:

`UNIF(nominal,relative_variation,[mult])`

Defines a uniform distribution. **PAR** can vary between `nominal - nominal*relative_variation` and `nominal + nominal*relative_variation`.

`AUNIF(nominal,absolute_variation,[mult])`

Defines a uniform distribution. **PAR** can vary between `nominal - absolute_variation` and `nominal + absolute_variation`.

`GAUSS(nominal,relative_variation,sigcoef,[mult])`

Defines Gaussian distribution. The standard deviation is equal to `relative_variation*nominal/sigcoef`.

`AGAUSS(nominal,absolute_variation,sigcoef,[mult])`

Defines Gaussian distribution. The standard deviation is equal to `absolute_variation/sigcoef`.

Note

The following two distribution statements are equivalent:

```
.PARAM P1=AGAUSS(2,0.3,3)
.PARAM P1=AGAUSS(2,0.6,6)
```

In both cases, the standard deviation will be 0.1.

`LIMIT(nominal,absolute_variation)`

Outputs **PAR** `- absolute_variation` or **PAR** `+ absolute_variation` depending on whether the random number, varying between -1 and 1, is negative or positive.

`nominal`
The nominal value.

`absolute_variation`
Absolute value for variation of the nominal value.

`relative_variation`
Relative value for variation of the nominal value.

`sigcoef`
Normalization coefficient.

`mult`
A positive integer value that acts as a multiplier to set how many times the parameter value is to be calculated. If it is greater than one then the distribution will be bimodal. The result could be either greater or lesser than the `nominal` value. The result with the largest deviation is then used. If `mult` is not specified it will default to 1.

Note

Parameter values can be specified with the percentage sign %. For example, the following two lines of syntax are equivalent:

```
.param rgauss=gauss(2,20%,1)
.param rgauss=gauss(2,0.2,1)
```

- **DEVX**=VAL

  The **DEVX** specification forces Eldo to use a new random value for each instance of a subcircuit. The difference with **DEV** is that even if a parameter is used several times in the same subcircuit, only one value will be used for that particular instance.

Note

DEVX can only be applied on **.PARAM**, unlike **LOT** and **DEV** which can be applied on both model parameters and **.PARAM** statements.

Note

It is impossible to have **DEV** and **DEVX** specified for the same parameter. If **DEV** and **DEVX** are specified for the same parameter, the last specification will be retained.

Related Options

PARAM_BEFORE_USE (see “PARAM_BEFORE_USE[=0|1]” on page 956), **USEFIRSTDEF** (see “USEFIRSTDEF” on page 962)

Examples

The following example shows how component values may be defined globally using the **.PARAM** command. Note that **LOT** and **DEV** values of 5% and 10% are also assigned to the parameter `lval` or MC analysis.

```
r1 1 2 rval
c1 1 2 cval
l1 1 2 lval
.param rval=2k cval=3p lval=2u lot=5% dev=10%
```

The following shows how parameters in a **.MODEL** definition may be assigned symbols which are then declared globally using the **.PARAM** command.

```
.model mod1 nmos level=3 vto=vtodef
*main circuit
m1 1 2 3 4 mod1 w=wdef l=ldef
.param vtodef=1 wdef=20u ldef=3u
```

The following shows arithmetic expressions and previously defined parameters combined in a **.PARAM** command.

```
r1 1 2 p2
.param p1=1k p3=2*p1
.param p2=sqrt(p1)+3*p3
```

The following example shows how parameters may be assigned symbols in a **.SUBCKT** definition. The parameters may then be given values explicitly when the subcircuit is called or globally using the **.PARAM** command.

```
*SUBCKT definition
.subckt inv 1 2
r1 1 3 rval
r2 3 4 rval1
r3 4 2 rval2
.ends inv
*subcircuit call
x1 1 2 inv rval=3 rval2=10
.param rval1=2
```

In the next example, the model name is substituted by the parameter `pmod`.

```
.param pmod="pmos1"
m1 d g s b $(pmod) w=1u l=1u
```

The following two examples show how string parameters are used.

```
.param MOD="Pmos1"
m1 d g s b $(MOD) w=1u l=1u

.param STIMFILE='"Stim.txt"'
v1 1 0 pwl file=$(STIMFILE)  R
```

The following example shows **LOT** and **DEV** specifications, defined for the **.param** and **.model** commands:

```
.param p1=10k lot=(UNI,-5%,4%, bi, 3%, 4%)
.model QND NPN BF=100 dev=(NOR,5%,5%,bi,-2%,2%, off, 1%)
```

The following example shows how a user-defined function can be specified. In this case, the value produced for **R1** would be 6 ohms.

```
.param rval(a,b)=a+b
R1 1 2 'rval(2,4)'
```

In the next example the **DEVX** declaration, placed on parameter `pc`, indicates that during a Monte Carlo analysis, a new value of `pc` has to be randomly generated for `X1`, and another one has to be generated for `X2`:

```
.param pc=10p DEVX=10%
.SUBCKT cmod a b
C1 a b pc
C2 a b pc
.ENDS
X1 4 0 cmod 10p
X2 6 8 cmod 10p
```

`X1.C1` and `X1.C2` will use the same value (same for `X2.C1` and `X2.C2`).

```
v1 1 0 dc 1 pwl (0 1 10n 9)
r1 1 0 1
v2 2 0 dc 1
```

```
r2 2 3 r={p2} tc1=4.2
r3 3 0 1

.param p1=v(1)
.param p2=2*p1

.tran 1n 10n
.extract tran yval(v(3),4n)
.plot tran v(3)
.end
```

This example shows how parameter `p1` can depend on the voltage at `v(1)`.

The following example will return the linearly interpolated value of p2 (y value) at an x value of p1.

```
.param p1=1
.param p2=pwl(p1, 1, 0, 10, 0.5, 20, 1.5 ,30)
```

The value of p2 will be 25, because the linear interpolation between the 2nd and 3rd pairs of values gives 25.

If linear interpolation was not specified, then p2=20. For more information, please refer to "PWL(xvalue, interp, x1, y1, ... xn, yn)" on page 79.

The following example shows **LOT** and **DEV** variation usage on MC distribution parameters.

```
.PARAM p1=UNIF(1,0.05)

.MODEL MOD1 NMOS VTO='1*p1'
.MODEL MOD2 PMOS VTO='-1*p1'

M1 ... W='p1*1u'
M2 ... W='p1*1u'
```

The same random value (**LOT** variation) will be used for the calculation of **VTO** for both **NMOS** and **PMOS** models, since in this case **p1** is affecting model parameters. However independent random values (**DEV** variation) will be used for the MOS instances **M1** and **M2**, since **p1** is affecting an instance parameter. Use option **PODEV** for backward compatibility for the mechanism in versions of Eldo prior to v6.3_2 where **DEV** variation was used for both model and instance parameters.

-compat flag

In `-compat` mode, **LOT** and **DEV** are not considered as keywords, however **LOT/GAUSS**, **DEV/GAUSS**, **LOT/**<distrib_name>, and **DEV/**<distrib_name> are.

In `-compat` mode, double quotes are considered as single quotes. (In standard Eldo mode, double quotes are used to specify a parameter string.) Use option **QUOTSTR** to consider double quotes as a parameter string delimiter.

In `-compat` mode, Eldo will check whether there is a **.model** with the same name as the string after the nodes in a model declaration. If not, it will look for a parameter name. This can be confusing, because if there are both a **.model** and a parameter name with the same name, then the simulator will consider the string to be a model name, while a parameter name was desired. This can be overcome by placing the parameter name in single quotes in the instantiation.

Please refer to “HSPICE Compatibility” on page 1373 for further information on the `-compat` flag.

In this example a parameter, `rmin`, is required as the value of a resistor instantiation. It is placed in single quotes so that it is viewed as a parameter and not a model name.

```
.model rmin fmax=3 nonoise
.param rmin=2k
r1 3 2 ‘rmin’
```

.PARAMDEX

Statistical Parameter Declarations

Noise Factors

```
.PARAMDEX FACTOR_NAME [NOISE/] SIG=NSIGMA
.PARAMDEX FACTOR_NAME [NOISE/] RNG=NRANGE
.PARAMDEX FACTOR_NAME [NOISE/] ABS=DELTA
.PARAMDEX FACTOR_NAME [NOISE/] REL=DELTA%
```

Designable Factors

```
.PARAMDEX FACTOR_NAME [CTRL/] ABS=DELTA [,NOM=RVALUE]
.PARAMDEX FACTOR_NAME [CTRL/] REL=DELTA% [,NOM=RVALUE]
```

Used to assign values to parameters (*factors*) used in design of experiments. The designer's selection of factors is accomplished by adding minor modifications to the working netlist, using the **.PARAMDEX** command.

A factor is a circuit parameter that is studied in the experiment. In order to study the effect of a factor on one or several user-defined responses, two or more values of the factor are used. These values are referred to as *levels* or *settings*. A combination of factor levels is called a *run* and will be associated to a specific simulation run in Eldo.

For the complete description, see the separate chapter Statistical Experimental Design and Analysis.

.PARAMOPT

Optimization Parameter Declarations

```
.PARAMOPT VARIABLE_NAME=(
+ [INIT_VALUE,]
+ {LOWER_BOUND | LOWER_PERCENT% },
+ {UPPER_BOUND | UPPER_PERCENT% }
+ [, INCREMENT])
```

The specification of the optimization variables is realized with this extension of the **.PARAM** command.

For the complete description of optimization capabilities, see the separate chapter Optimizer in Eldo.

.PART

Circuit Partitioning

```
.PART ADIT|ELDO|MODSST
+ INST=(<instance_list>) SUBCKT=(<subcircuit_list>)
```

The **.PART** command instructs Eldo to use the specified algorithm in place of the regular transient algorithm, for a certain selection of instances. This performs circuit partitioning.

The instances to be simulated with the specified algorithm may be listed explicitly, using the <instance_list>. They may also be implicitly designated, using the <subcircuit_list>.

Parameters

The list below details how Eldo will partition the circuit, the keyword specified selects a simulation algorithm for the according instances/subcircuits:

- **ADIT**

  Use the ADiT algorithm.

- **ELDO**

  Use the Eldo algorithm.

- **MODSST**

  Use the Eldo RF MODSST algorithm. For further details, see .MODSST (Modulated Steady-State analysis) of the *Eldo RF User's Manual*.

- <instance_list>

  This is enclosed in parenthesis, and contains a list of instance names, separated by commas, possibly using wildcards (* and ?) in place of characters.

- <subcircuit_list>

  This is enclosed in parenthesis, and contains a list of subcircuit names separated by commas. Subcircuit names may also contain wildcard characters (* and ?). If a subcircuit appears in the <subcircuit_list>, all instances of this subcircuit will be handled with the specified algorithm.

Example

Using these directives will have an effect on how and where simulation occurs, for example:

```
.part ADIT SUBCKT= (NOR1, NAND3, AOIX*)
.part ADIT INST=(XA1.XM2)
.part ELDO INST=(XAMP.XAGC)
.part MODSST SUBCKT=(INV_SPICE)
```

.PLOT

Plotting of Simulation Results

```
.PLOT [ANALYSIS] OVN [(LOW, HIGH)] [(VERSUS)]
+ {OVN [(LOW, HIGH)]} [UNIT=NAME] [(SCATTERED)] [STEP=value]
.PLOT AC|FSST S(i, j) [(SMITH[,zref])] [(POLAR)]
.PLOT FOUR FOURxx(label_name) [(SPECTRAL)] [(CONTINUOUS)]
.PLOT DSP DSPxx(label_name)
.PLOT EXTRACT [MEAS(meas_name) | SWEEP(sweep_name)]
.PLOT [CONTOUR] MEAS(meas_name_x) MEAS(meas_name_y) [(SCATTERED)]
+ [(SMITH[,zref])] [(POLAR)]
.PLOT [ANALYSIS] TWO_PORT_PARAM [(SMITH[,zref])] [(POLAR)]
```

The **.PLOT** command takes its name from Berkeley SPICE (2G6). It is used to specify which simulation results have to be kept by the simulator for graphical viewing and post-processing. For the first syntax above, at least one OVN (output variable name) plot specification is required to plot values of specific nodes or components. See “Plot Specifications (OVN)” on page 799.

See full list of “Parameters” on page 796.

Files created by .PLOT Commands

The quantities listed in a **.PLOT** command are written to binary output files (*.wdb*). The binary files are the input data for the EZwave waveform viewer compatible with Eldo. Saved windows files *.swd* are created with a *.wdb* database and contain information on specific graph windows such as size or position. They also contain the waveforms associated with graph windows. When specifying plots for frequency based analysis, the *.swd* file contains the information for complex waveforms, which can then be viewed by opening the *.swd* file with EZwave.

Note

Specify option **NOWAVECOMPLEX** (see “NOWAVECOMPLEX” on page 1005) to force complex waveform information to be stored within the waveform database (*.wdb*) as well as the *.swd* file. This restores the functionality of Eldo versions prior to v6.3_2.1.

By default, the quantities listed in a **.PLOT** command are not written to the main ASCII output file (*.chi*), see “ASCII Outputs in the .chi File” on page 795.

Types of Waveforms

Many different simulation results can be created by Eldo. The simulator can output simple node voltages, but also currents through devices, currents through device or subcircuit pins, power quantities, S parameters, internal variables from device models, and so on. All these results are direct *raw* results from a simulation. The tables shown in the next few pages indicate the exact syntax to use for each category. Example:

```
.PLOT TRAN V(OUT)
.PLOT DC ISUB(XBIAS.VOUT)
```

The **.PLOT** command may also be used to plot *extraction* results. A flexible language exists in Eldo to *extract* characteristics from raw simulation results (for example the maximum value of a waveform, or the time at which a given threshold is crossed by a waveform, and so on). See “.EXTRACT” on page 637.

When extraction statements are combined with parameter sweeping statements (see “.STEP” on page 890), Eldo automatically creates waveforms showing the extraction results versus the swept parameter. For example, a user may extract the width of an output pulse from a transient simulation, and sweep the power supply level. In this case Eldo will automatically create a waveform showing the width of the pulse versus the power supply level. Similarly, if extractions and **.ALTER** statements are combined, Eldo will automatically create waveforms showing the extraction results versus the index of the **.ALTER** runs (see “.ALTER” on page 549). In this case, the X axis will contain 1, 2, 3, and so on. The initial display in the viewer can also be prepared using **.PLOT** commands (see the **.PLOT EXTRACT**... description, “EXTRACT” on page 798). Example:

```
.EXTRACT TRAN label=VMAX  MAX(V(out))
.PARAM powersupply=1.2
VDD VDD 0 'powersupply'
.STEP PARAM powersupply list 1.2V 1.3V 1.4V 1.6 2V
.PLOT EXTRACT meas(VMAX) ! this will create a waveform
*                          showing VMAX(powersupply)
```

By default, using the *.wdb* format, the **.EXTRACT** waveforms are saved inside the *EXT* folder in the main *.wdb* file. Specify option **EXTFILE** enables the generation of a *.ext.wdb* file with only the extraction results.

If using the *.cou* format, these **.EXTRACT** waveforms are created in a separate file with the *.ext* extension. The command-line switch **-couext** (**eldo -couext -i** <*netlist.cir*>) can be used to merge the extract waveforms into the *.cou* file, when possible. In this case, a single *.cou* file will contain the *raw* waveforms and also the extraction waveforms.

Finally, the **.PLOT** command, combined with the **.DEFWAVE** command (see “.DEFWAVE” on page 605) can also be used to generate so-called *template* waveforms, that is. piece-wise-linear waveforms defined by a series of arbitrary (x,y) coordinates. Using **.PLOT**, these templates can be displayed together with the simulation results (this is very useful when verifying whether a filter response passes or not specifications such as cutoff frequencies, minimum attenuation and so on). Example:

```
.DEFWAVE FILTERSPEC=PWL(1e-3,0,100k,0,500k,-60,100G,-60)
.PLOT AC VDB(OUT) W(FILTERSPEC)
```

Complex Modifiers and Initial Formatting

Some results are real quantities, such as the currents and voltages from a transient analysis, while others are complex. In this case, real or imaginary parts, magnitude or phase or group delay, can be required by the user.

It is also possible to specify some initial basic *formatting* of the results. For example, spectral or scattered display modes can be specified, instead of the default *continuous-line* mode. S, Y, Z parameters can be automatically displayed in a Smith chart, and so on. These optional specifications do not alter the raw data, they only tell the waveform viewer how to display the data initially.

X axis of Waveforms

In general, the created waveforms have their X axis implicitly defined by the corresponding analysis. For example, the X axis of a transient waveform is the time, the X axis of an AC waveform is the frequency and so on. The X axis for a transient waveform is defined by the **.TRAN** command, it ranges from `t=0` to `t=tmax` where `tmax` is specified in the **.TRAN** `tstep tmax` command (see ".TRAN" on page 911). The X axis for an AC waveform is defined by the frequency points in the **.AC** command (see ".AC" on page 538). DC and NOISE waveforms also inherit their X axis from the corresponding analysis command (**.DC** or **.NOISE**).

For all analyses but the transient analysis, the X-axis range and the spacing of points in the X axis are thus predictable. For example, the frequency points in a **.AC** command are either listed explicitly or regularly spaced, in a predictable way. For transient analysis however, this is different. Eldo always uses a variable timestep algorithm, and the spacing of the timepoints where the circuit is solved is dictated by accuracy considerations only (see the Speed and Accuracy chapter). Thus the total number of timepoints is generally not predictable. By default, all computed timepoints are stored in the binary output files, so that any rapid change or *glitch* can be inspected visually in the waveform processor. However, when working with large circuits and/or long transient simulations and/or storing many waveforms, this may generate huge output files. The loading and post-processing of these huge output files by the waveform viewers may thus be slower. In extreme cases, this may also impact the simulation time, as writing gigabytes to a disk, possibly through a busy network, may take quite a while. The next section discusses several options meant to control the size of the transient output files.

In the case of **.EXTRACT** waveforms, the X axis is defined by the parameter sweeping statement (**.STEP**), or it contains integer indexes (**.ALTER**).

Limiting the Size of Transient Output Files

At least three options are available in Eldo to limit the size of the output files for transient simulation. The first one is the option **OUT_STEP**=`val`, the second one is option **OUT_RESOL**=`resolution`, and the third one is option **INTERP**=`1`. Of course, the many options related to the accuracy settings do generally have an impact upon the number of computed points and thus upon the size of the output files, but this is only qualitative and indirect. There is no way to quantitatively relate, say the *eps* or the *reltol* specification, to the final size of the output files—this all depends on the circuit. In contrast, the options discussed in this section allow controlling the amount of data in a predictable way (the **OUT_RESOL** option only sets a maximum size, whereas **OUT_STEP** and **INTERP** allow exact predictions). These options may have a considerable impact on the overall simulation speed, so it is important to define exactly what has to be achieved.

- Option **OUT_STEP**

  The **.OPTION OUT_STEP**=val command (see “OUT_STEP” on page 1006) forces a timepoint at each multiple of val, and only these timepoints are stored in the binary output files. These timepoints are forced, that is, they come in addition to the normal timepoints picked by the simulator. When using this option, the timepoints are forced, that is, they are computed even if they are not required from an accuracy point of view. One of the applications of the **OUT_STEP** option is for FFT computations. In this case, it is frequent that the user wants to have exact, computed points, at regularly spaced timepoints, to minimize the interpolation artifacts when computing an FFT. However the **OUT_STEP** option is also a way to control the size of the output files, even if no FFT is scheduled. If not chosen properly, a possible risk of the **OUT_STEP** option is to slowdown the simulation unnecessarily, by forcing Eldo to compute more points than needed. This is particularly true if they are long periods of time with little or no activity in the simulation. Forcing a short *out_step* will force many useless timepoints during these periods, whereas Eldo would normally accelerate and compute fewer timepoints. This option is however preferred by some users because the output waveforms ultimately contain only computed points, without interpolation.

- Option **OUT_RESOL**

  Another option used to reduce the size of the output file is the **OUT_RESOL** option. With this option, the user can specify the smallest *resolution* of the output file (see “OUT_RESOL=VAL” on page 1006). Computed data is then dumped only if the current time is greater than the previously written timepoint, augmented by the resolution. For example if **OUT_RESOL** is set to 1ns, and the simulator has written data at time t=24.65ns, all timepoints computed until t=25.65ns will not be dumped. The first timepoint after t=25.65ns will be dumped. This could be t=25.76ns for example, or t=27ns if the simulator can *accelerate* in this period of time. No timepoints are forced other than those naturally picked by the simulator. Thus the spacing of timepoints in the output file is generally non-constant. The maximum average density of timepoints in the output file is determined by the resolution parameter of the option. The minimum density is not guaranteed, neither locally, nor globally. This option is an interesting alternative, combining the certainty that the points in the output are computed points only (no interpolation error), and the ability to handle low activity periods efficiently. It is however, generally not well suited for simulations where an FFT must be computed, because of the non-constant spacing of the timepoints (this would result in interpolation errors when re-sampling for the FFT).

- Option **INTERP**

  Both **OUT_STEP** and **OUT_RESOL** options dump only computed points to the output file. Sometimes it may be desirable to obtain equally spaced time points, but forcing timepoints (with **OUT_STEP** for example) would result in an unacceptable CPU penalty. In these cases, interpolating may be useful. An interpolation option is available to force Eldo to sample the output data along the <tstep> parameter of the **.TRAN** command (see “.TRAN” on page 911). If using the option **INTERP**=1 (see “INTERP” on page 978), the transient output waveforms are interpolated, and the timepoints written to the file are aligned with multiples of `<tstep>` exactly. For example, if using both the following:

```
.tran 1ns 100ns
.option interp=1
```

the output waveforms will contain exactly 101 points (time 0 is always included), spaced by 1ns. Glitches shorter than 1ns may not be caught in the resulting waveform. This is a radical way to reduce the size of the binary output files (*.wdb*). However, the written data is obtained by interpolation, thus including an unpredictable amount of interpolation error. During low-activity periods, Eldo still picks the largest possible timesteps which will still allow to meet the accuracy requirements, and *fills* the output data with interpolated data (at no or little cost, as opposed to the **OUT_STEP** technique discussed previously).

ASCII Outputs in the .chi File

By default, the quantities listed in a **.PLOT** command are not written to the ASCII *.chi* output file. This avoids creating huge *.chi* output files in the case of large simulations. The ASCII outputs are still supported for SPICE compatibility reasons, although seldom used. To force Eldo to generate these output quantities in the ASCII output file, specify the `-ascii` command-line flag when invoking Eldo or include the option **ASCII** in the netlist (see "ASCII" on page 998). Symbols such as *, +, and so on, will be used to identify each quantity, to create ASCII *waveforms* in the *.chi* output file.

Note

For transient results, the spacing of these points in the ASCII *waveforms* is defined by the `<tstep>` parameter of the **.TRAN** command (data is interpolated). This is for original SPICE compatibility. For all other analyses, the points in the X axis are those specified by the corresponding analysis command.

Wildcards

You can include wildcards in **.PLOT** commands. See "Using Wildcards in Subcircuit Instances" on page 843 for more information on using wildcards with **.PLOT**/**.PROBE** commands.

Waveforms specified in previous **.PLOT** commands are excluded from the wildcard plot.

Related Commands

See the **.PRINT** and **.PROBE** commands.

Parameters

- `ANALYSIS`

  Optional. Specifies the analysis type, which can be useful in the case of multiple types of analysis in the *.cir* file. Can be one of the following:

  DC
  Specifies that the plots are required for a DC analysis.

  AC
  Specifies that the plots are required for an AC analysis.

  TRAN
  Specifies that the plots are required for a transient analysis.

  NOISE
  Specifies that the plots are required for a noise analysis.

  SSTAC|SSTXF|SSTNOISE|SSTJITTER|TMODSST|FMODSST|FOURMODSST|TSST|FSST|SSTSTABIL
  Please refer to the Display Command Syntax chapter of the *Eldo RF User's Manual* for more information regarding these RF options.

  OPT
  Extraction during optimization analysis. Can only be used with wave type **WOPT**. This wave type is used for optimization analysis only, that is, it may only be used with **OPT**. For more information on wave type **WOPT**, see "WOPT" on page 1191.

- **FOUR**

  Displays FFT results. Please see `FOURxx(label_name)` below for display options.

- **DSP**

  Displays DSP results. Please see `DSPxx(label_name)` below for display options. See also ".DSP" on page 615.

- `OVN`

  Output variable name. Requests plotting of values of specific nodes on components. Required. The syntax for specifying the list of plot specifications to be monitored is under "Plot Specifications (OVN)" on page 799.

- **FOUR**`xx(label_name)`

  Should be specified as part of `OVN`. Displays FFT results. `xx` stands for **DB**, **M**, **P**, **R**, **I**:

  DB Magnitude, in dB
  M Magnitude
  P Phase
  R Real part
  I Imaginary part

- **DSP**`xx(label_name)`

  Should be specified as part of `OVN`. Displays DSP results. `xx` stands for **DB**, **M**, **P**, **R**, **I**, **GD** (see above), **GD** is Group Delay.

- `LOW, HIGH`

  The optional plot limits `LOW` and `HIGH` may be specified for each of the output variables. All output variables of the same kind (voltage for instance) to the left of a pair of plot limits (`LOW, HIGH`) will be plotted using the same lower and upper bounds. If plot limits are not specified, Eldo uses the following default values:

  - 0V and 5V for `LOW` and `HIGH` values respectively in voltage plots
  - −100dB and 100dB for `LOW` and `HIGH` values respectively in dB plots
  - −180 and 180 degrees for `LOW` and `HIGH` values respectively in phase plots

- **`UNIT`**`=NAME`

  Displays user-defined text representing the Y-axis units of a plot. Useful in conjunction with the **`.DEFWAVE`** command and with Eldo-FAS.

- (**`VERSUS`**)

  Specifies the waveform viewer plot one, or a number of waves on the Y-axis against a single wave on the X-axis. (**`VERSUS`**) must be preceded by at least one wave and followed by one wave only.

- `(`**`SCATTERED`**`)`

  Only computed points will be represented (no "line" between two successive points). This property is active for all waves of the graph.

- `STEP=value`

  Performs a sampling of the waveform(s). A point is dumped every `value`. It can only be specified for databases that support multiple x-axes in the same simulation, for example JWDB. Other databases will ignore this parameter.

- **`S`**`(i, j)`

  Smith chart specification of the S parameter S_{ij}.

- `(`**`SMITH`**`[,zref])`

  Prompts the waveform viewer to display the waves which are given in the corresponding **`.PLOT`** command in a Smith chart. Waves must be given without any AC extension, otherwise Eldo will issue an error. The **`SMITH`** keyword can only be used in conjunction with Smith chart specifications above. `zref` is optional and specifies the impedance.

- `(`**`POLAR`**`)`

  Prompts the waveform viewer to display the waves which are given in the corresponding **`.PLOT`** command in a Polar chart.

- `(`**`SPECTRAL`**`)`

  Specifies the waveform viewer to represent the wave in its spectral representation. This is the default for FFT waveforms. This keyword must be placed at the end of the command, and be enclosed in parentheses.

- (**CONTINUOUS**)

  Specifies the waveform viewer to represent the wave in continuous mode instead of the default spectral representation. This keyword must be placed at the end of the command, and be enclosed in parentheses. This modifies the *.swd* file only; FFT waveforms are still saved in the *.wdb* file as spectral. Can be alternatively specified with the option **CONTINUOUS_FFT**.

- **EXTRACT**

  Allows plot combinations in *.ext* file. This is to be used in conjunction with **.ALTER**/**.STEP**/**.TEMP** to organize extracted waves in graphs. For example, if you have:

  ```
  .EXTRACT LABEL = a <expression>
  .EXTRACT LABEL = b <expression>
  ```

  use **.PLOT EXTRACT** MEAS(a) MEAS(b) if both waves are expected in the same plot on the waveform viewer invocation.

- **CONTOUR**

  Eldo will plot the second measurement value with respect to the first. The wave will be displayed in a Smith chart if keyword **SMITH** is specified. **CONTOUR** information is written to the *.ext* file. At least one **.STEP** command is required to use Contour plots.

Note

The keyword **VECT** or **CATVEC** must have been specified on the **.EXTRACT** which is referred to in the **.PLOT CONTOUR**, otherwise the corresponding data will be reduced to a single print.
The two measurements in the **CONTOUR** must have the same dimension. If not, the **.PLOT CONTOUR** will be ignored.

See the Contours section of the *Eldo RF User's Manual* for more information on using **.PLOT CONTOUR**.

- TWO_PORT_PARAM

  This must be replaced by one of the following keywords:

Table 10-23. Two-port Parameter Keywords

BFACTOR	BOPT	GA_mag	GA_dB	GAC
GAM_mag	GAM_dB	GAMMA_OPT	GAMMA_OPT_MAG	GASM_mag
GASM_dB	GAUM_mag	GAUM_dB	GOPT	GP_mag
GP_dB	GPC	KFACTOR	LSC	MUFACTOR
MUFACTOR_L	MUFACTOR_S	NC	NFMIN_mag	NFMIN_dB
PHI_OPT	RNEQ	SSC	TGP_mag	TGP_dB
YOPT				

Note

All of these keywords may appear in expressions. However, they may not be specified in a **.PARAM** command if an RF analysis is specified in the netlist. These keywords are all available with both AC and FSST results.

Plot Specifications (OVN)

- **NOISE**(element_name)

  Should be specified as part of OVN. Specifies that the plots are required for a noise analysis of a specific component or subcircuit instance. Multiple occurrences of this parameter can appear in a single line of syntax.

- **INOISE**

  Should be specified as part of OVN. Input noise when performing a noise analysis.

- **ONOISE**

  Should be specified as part of OVN. Output noise when performing a noise analysis.

Note

NOISE(element_name), **INOISE** or **ONOISE** can appear singularly or as part of any combination. It is necessary that at least one be specified.
INOISE and **ONOISE** may appear in expressions. However, they may not be specified in a **.PARAM** command.

For more information on the use of **INOISE** and **ONOISE**, see the example in ".NOISE" on page 744 and "ONOISE" on page 813.

- FOURxx(label_name)

  Displays FFT results. Should be specified as part of OVN. xx can be **DB**, **M**, **P**, **R**, **I**, **GD**:

  DB Magnitude, in dB
  M Magnitude
  P Phase
  R Real part
  I Imaginary part
  GD Group Delay

- DSPxx(label_name)

  Displays DSP results. Should be specified as part of OVN. xx can be **DB**, **M**, **P**, **R**, **I**, **GD** (see meanings above).

- LSTB_xx

  Displays loop stability analysis (LSTB) results. Should be specified as part of OVN. xx can be **DB**, **M**, **P**, **R**, **I**, **GD** (see meanings above).

- **VDIG**(node_name)

  Should be specified as part of OVN. Enables the plotting of an analog signal as a digital bus. See "Plotting an analog signal as a digital bus" on page 823 for further details.

- **WX**xx(devname.posi)

  Should be specified as part of OVN. Returns the value of complex waveforms defined by the **.DEFWAVE** command or implicit declarations such as **W**('P(parameter_value)') and **W**('wave_dependent_expression'). **WX**xx can be: **W**, **WDB**, **WP**, **WR**, **WI**, **WM**, **WGD**, or **WOPT**.

- **VX**xx**(**devname.posi**)**

  Should be specified as part of OVN. Returns the voltage on the pin. **VX**xx can be: **VX**, **VXDB**, **VXP**, **VXR**, **VXI**, **VXM**, or **VXGD**.

- **IX**xx**(**devname.posi**)**

  Should be specified as part of OVN. Returns the current on the pin. **IX**xx can be: **IX**, **IXDB**, **IXP**, **IXR**, **IXI**, **IXM**, or **IXGD**. The current is positive when it enters the object by this pin.

  devname

  Device name.

  posi

  Position of the pin as given in the *.cir* file. posi can be greater than the number of external pins. In such a case, internal pin values are given. Internal pins are sorted in the same order as external pins (5 on a BJT for instance, refers to the 5th pin, which is the 1st internal pin, which is the internal collector). The case for GUDM models (Mextram, HICUM) is different, because the intrinsic structure of such models is not known to Eldo. The internal pins can be sorted in any order, therefore the value of intrinsic pins for GUDM models can be accessed only by the writer of the model.

- **DATA**(dataname,parameter)

  Should be specified as part of OVN. Allows to plot a wave defined with a **.DATA** statement. dataname is the name assigned to the **.DATA** statement. It is mostly used to define fitting waves during optimization.

- **IPROBE**(vname)

  Should be specified as part of OVN. Returns the sum of the currents on the **.IPROBE** zero voltage source. See ".IPROBE" on page 689.

- **IN(**nodename**)**

  Should be specified as part of OVN. Expands the command into multiple plot instructions corresponding to the objects connected to that pin. For example:

  ```
  R1 1 2 1
  R2 2 0 1
  .PLOT tran in(2)
  ```

  the plot specification will expand the plot into two items:

```
.PLOT tran ix(r1.2) ix(r2.1)
```

- **INX(**`Xinstance.index`**)**

  Should be specified as part of `OVN`. Expands the command into multiple plot instructions corresponding to the devices connected to the instance at the specified pin. For example:

```
.SUBCKT foo a b
r1 a b 1
r2 b 0 1
.ends
X1 a b foo

.PLOT TRAN INX(X1.2)
```

  the plot specification will expand the plot into two items:

```
.PLOT tran ix(x1.r1.2) ix(x1.r2.1)
```

Note

These two command parameters `IN()` and `INX()` can appear in **.PLOT**, **.PROBE**, or **.PRINT** exclusively. This is because `IN(node)` is always 0, (the sum of the currents flowing onto a pin is 0), and if you want to access current entering a subckt by a specific pin, then command `IX(instance.index)` is already provided and should be used instead of `INX()`.

Note

When the `-compat` flag is set, the following apply:
For R/L/C/E/F/G/H/I/V/D devices, **I**`2` returns the same value as **I**`1`.
For M/B/J devices, **I**`3` is positive when current leaves the object by pin number 3.

- **OPMODE(**`device_name`**)**

  Should be specified as part of `OVN`. Returns the DC working mode of the device model, `device_name`. Within **.PLOT**/**.PROBE** the use of **OPMODE** is restricted to JWDB files because this is the only output format supporting enumerated waveforms (the wave **OPMODE()** is represented as a digital waveform using states linear/saturation/subthreshold, etc.). Eldo plots the DC working mode of a device as a string instead of a real value. Strings returned are:

  - for MOS devices:
    LINEAR, if *VGS* > *VT* and *VDS* < *VDSAT*
    SATURATION, if *VGS* > *VT* and *VDS* > *VDSAT*
    SUBTHRESHOLD, if *VGS* < *VT*
  - for BJT devices:
    SATURATION, if *VBE* > 0 and *VBC* > 0
    ON, if *VBE* > 0 and *VBC* < 0
    OFF, if *VBE* < 0 and *VBC* < 0
    INVERSE, if *VBE* < 0 and *VBC* > 0

The following example plots returns the DC working mode of device `m2` in subcircuit instances `x1` and `x2`.

```
.plot tran opmode(x2.m2)  opmode(x1.m2)
```

Below is a list of devices and the syntax used to plot or print the values of both their extrinsic and intrinsic pins. The information is divided first by device and then into subsections of output type and analysis type. For devices, if the position number specified is greater than the index available for the device, it will return a value of zero.

The plot specifications must match the corresponding analysis type, **AC**, **DC**, **TRAN**, or **NOISE**.

For examples of syntax usage, see the "Examples" on page 823.

The output quantities listed in Table 10-24 are generic for MOSFET models. Some newer models have names that are different than listed here. These are documented with the model, as is the case for the BSIM4 model.

Table 10-24. MOSFET Plotting and Printing

DC or Transient Analysis	
Currents	
ID(Mxx)/I(Mxx.D)/IX(Mxx.1)	Drain current
IG(Mxx)/I(Mxx.G)/IX(Mxx.2)	Gate current
IS(Mxx)/I(Mxx.S)/IX(Mxx.3)	Source current
IB(Mxx)/I(Mxx.B)/IX(Mxx.4)	Bulk current
IBD(Mxx)	Bulk-drain diode current
IBS(Mxx)	Bulk-source diode current
Voltages	
VD(Mxx)/VX(Mxx.1)	Drain voltage, in Volts
VG(Mxx)/VX(Mxx.2)	Gate voltage, in Volts
VS(Mxx)/VX(Mxx.3)	Source voltage, in Volts
VB(Mxx)/VX(Mxx.4)	Bulk voltage, in Volts
VGS(Mxx)	Gate-source voltage, in Volts
VGB(Mxx)	Gate-bulk voltage, in Volts
VDS(Mxx)	Drain-source voltage, in Volts

Table 10-24. MOSFET Plotting and Printing

VBS(Mxx)	Bulk-source voltage, in Volts
VBD(Mxx)	Bulk-drain voltage, in Volts
VT(Mxx)	Threshold voltage value, in Volts
VDSS(Mxx)	Saturation voltage value, in Volts
Conductances	
GDS(Mxx)	$\frac{\partial I_D}{\partial V_{DS}}$
GM(Mxx)	Transconductance: $\frac{\partial I_D}{\partial V_{GS}}$
GMBS(Mxx)	$\frac{\partial I_D}{\partial V_{BS}}$
GMIBD(Mxx)	Mosfet diode drain conductances
GMIBS(Mxx)	Mosfet diode source conductances
Capacitances	
CGS(Mxx)	$\frac{\partial QG}{\partial Vs}$ Gate/Source capacitance
CGD(Mxx)	$\frac{\partial QG}{\partial Vd}$ Gate/Drain capacitance
CGG(Mxx)	$\frac{\partial QG}{\partial Vg}$ Gate/Gate capacitance
CGB(Mxx)	$\frac{\partial QG}{\partial Vb}$ Gate/Bulk capacitance
CBS(Mxx)	$\frac{\partial QB}{\partial Vs}$ Bulk/Source capacitance
CBD(Mxx)	$\frac{\partial QB}{\partial Vd}$ Bulk/Drain capacitance
CBG(Mxx)	$\frac{\partial QB}{\partial Vg}$ Bulk/Gate capacitance

Table 10-24. MOSFET Plotting and Printing

CBB(Mxx)	$\frac{\partial QB}{\partial Vb}$ Bulk/Bulk capacitance
CBSJ(Mxx)	$\frac{\partial QB}{\partial Vbs}$ Bulk/Source junction diode capacitance
CBDJ(Mxx)	$\frac{\partial QB}{\partial Vbd}$ Bulk/Drain junction diode capacitance
CDS(Mxx)	$\frac{\partial QD}{\partial Vs}$ Drain/Source capacitance
CDD(Mxx)	$\frac{\partial QD}{\partial Vd}$ Drain/Drain capacitance
CDG(Mxx)	$\frac{\partial QD}{\partial Vg}$ Drain/Gate capacitance
CDB(Mxx)	$\frac{\partial QD}{\partial Vb}$ Drain/Bulk capacitance
CSS(Mxx)	$\frac{\partial QS}{\partial Vs}$ Source/Source capacitance
CSD(Mxx)	$\frac{\partial QS}{\partial Vs}$ Source/Drain capacitance
CSG(Mxx)	$\frac{\partial QS}{\partial Vg}$ Source/Gate capacitance
CSB(Mxx)	$\frac{\partial QS}{\partial Vb}$ Source/Bulk capacitance
CGDO(Mxx)	Gate/Drain overlap capacitance
CGSO(Mxx)	Gate/Source overlap capacitance
CGBO(Mxx)	Gate/Bulk overlap capacitance
Charges[1]	
QG(Mxx)	Gate charge

Table 10-24. MOSFET Plotting and Printing

QB(Mxx)	Bulk charge
QD(Mxx)	Drain charge
QS(Mxx)	Source charge
QBD(Mxx)	Bulk-drain junction charge
QBS(Mxx)	Bulk-source junction charge
Power	
POW(Mxx)	Power curve calculated as the product of the drain to source voltage and the drain to source current (Vds $\times$ Ids)
AC Analysis	
Currents[2]	
Izz(Mxx.D)	Drain current. See below for meaning of zz
Izz(Mxx.G)	Gate current. See below for meaning of zz
Izz(Mxx.S)	Source current. See below for meaning of zz
Izz(Mxx.B)	Bulk current. See below for meaning of zz
Noise Analysis	
IBSNOISE(Mxx)	Base-Source diode noise: shot noise due to IBD
IBDNOISE(Mxx)	Base-Drain diode noise: shot noise due to IBS
RSNOISE(Mxx)	Source access resistor noise
RDNOISE(Mxx)	Drain access resistor noise
RGNOISE(Mxx)	Gate access resistor noise (for MOS model with RG resistance)
THNOISE(Mxx)	Thermal noise
FLKNOISE(Mxx)	Flicker noise
Instance Parameters	
EVAL(Mxx, Leff)	Effective length
EVAL(Mxx, Weff)	Effective width
EVAL(Mxx, ADeff)	Effective drain area
EVAL(Mxx, ASeff)	Effective source area
EVAL(Mxx, PDeff)	Effective drain perimeter
EVAL(Mxx, PSeff)	Effective source perimeter
EVAL(Mxx, Geo)	The GEO parameter (geometry selector)

Table 10-24. MOSFET Plotting and Printing

EVAL(Mxx, RDeff)	Effective drain series resistance
EVAL(Mxx, RSeff)	Effective source series resistance
Special Outputs in the .chi File	
VTH_D	Vgs - Vth

1. Available for charge controlled models only.
2. In the above specifications, zz is to be replaced by one of the following:
DB (Magnitude, in dB); M (Magnitude); P (Phase); R (Real part); I (Imaginary part); GD (Group delay).

Table 10-25. BJT Plotting and Printing

DC or Transient Analysis	
Currents	
IC(Qxx)/I(Qxx.C)/I(Qxx.1)	Collector current
IB(Qxx)/I(Qxx.B)/I(Qxx.2)	Base current
IE(Qxx)/I(Qxx.E)/I(Qxx.3)	Emitter current
IS(Qxx)/I(Qxx.S)/I(Qxx.4)	Substrate current
Voltages	
VBEI(Qxx)	Internal node voltage difference base-emitter
VBCI(Qxx)	Internal node voltage difference base-collector
VCEI(Qxx)	Internal node voltage difference collector-emitter
VBE(Qxx)	External node voltage difference base-emitter
VBC(Qxx)	External node voltage difference base-collector
VCE(Qxx)	External node voltage difference collector-emitter
Conductances	
GM(Qxx)	Transconductance: $\frac{\partial I_C}{\partial V_{BE}}$
Capacitances	
CBS(Qxx)	Base/Substrate capacitance (LPNP only)
CBSX(Qxx)	Extrinsic Base/Substrate capacitance (LPNP only)
CBX(Qxx)	Extrinsic Base/Collector capacitance
CCS(Qxx)	Collector/Substrate capacitance
CMU(Qxx)	Base/Collector capacitance
CPI(Qxx)	Base/Emitter capacitance

Table 10-25. BJT Plotting and Printing

Resistances	
RO(Qxx)	Resistance between internal nodes C and E
RPI(Qxx)	Resistance between internal nodes B and E
RX(Qxx)	Series resistance between internal and external Base nodes (Base Resistance)
RBB(Qxx)/RX(Qxx)	Base resistance value
Power	
POW(Qxx)	Power curve calculated as the product of the collector to emitter voltage and the collector to emitter current (VCE × ICE)
Frequency	
FT(Qxx)	Frequency at which small-signal forward current transfer ratio extrapolates to unity
AC Analysis	
Currents[1]	
Izz(Qxx.C)	Collector current. See below for meaning of zz
Izz(Qxx.B)	Base current. See below for meaning of zz
Izz(Qxx.E)	Emitter current. See below for meaning of zz
Izz(Qxx.S)	Substrate current. See below for meaning of zz
Noise Analysis	
ICNOISE(Qxx)	Shot IC noise
IBNOISE(Qxx)	Shot IB noise
THNOISE(Qxx)	Thermal noise
FLKNOISE(Qxx)	Flicker noise
RBNOISE(Qxx)	Thermal noise due to access resistance RB
RENOISE(Qxx)	Thermal noise due to access resistance RE
RCNOISE(Qxx)	Thermal noise due to access resistance RC

1. In the above specifications, zz is to be replaced by one of the following:
DB (Magnitude, in dB); M (Magnitude); P (Phase); R (Real part); I (Imaginary part); GD (Group delay).

Table 10-26. JFET Plotting and Printing

DC or Transient Analysis	
Currents	
ID(Jxx)/I(Jxx.D)/I(Jxx.1)	Drain current
IG(Jxx)/I(Jxx.G)/I(Jxx.2)	Gate current
IS(Jxx)/I(Jxx.S)/I(Jxx.3)	Source current
Voltages	
VD(Jxx)/V(Jxx.D)/V(Jxx.1)	Drain voltage, in Volts
VG(Jxx)/V(Jxx.G)/V(Jxx.2)	Gate voltage, in Volts
VS(Jxx)/V(Jxx.S)/V(Jxx.3)	Source voltage, in Volts
VGS(Jxx)	Gate-source voltage, in Volts
VDS(Jxx)	Drain-source voltage, in Volts
VT(Jxx)	Threshold voltage value, in Volts
VDSS(Jxx)	Saturation voltage value, in Volts
Conductances	
GDS(Jxx)	$\frac{\partial I_D}{\partial V_{DS}}$
GM(Jxx)	Transconductance: $\frac{\partial I_D}{\partial V_{GS}}$
Capacitances	
CGD(Jxx)	$\frac{\partial QG}{\partial Vd}$ Gate/Drain capacitance
CGS(Jxx)	$\frac{\partial QG}{\partial Vs}$ Gate/Source capacitance
AC Analysis	
Currents[1]	
Izz(Jxx.D)	Drain current. See below for meaning of zz
Izz(Jxx.G)	Gate current. See below for meaning of zz
Izz(Jxx.S)	Source current. See below for meaning of zz

Table 10-26. JFET Plotting and Printing

Noise Analysis	
RGNOISE(Jxx)	Gate access resistor noise (for JFET MODEL with RG resistance)

1. In the above specifications, zz is to be replaced by one of the following:
DB (Magnitude, in dB); M (Magnitude); P (Phase); R (Real part); I (Imaginary part); GD (Group delay).

Table 10-27. Diode Plotting and Printing

DC or Transient Analysis	
Currents	
IX(Dxx.1)/I(Dxx.POS)/ ID(Dxx)/I(Dxx)	Positive output
IX(Dxx.2)/I(Dxx.NEG)	Negative output
Conductances	
GD(Dxx)	Diode conductance
Capacitances	
CD(Dxx)	Diode capacitance
Noise Analysis	
FLKNOISE(Dxx)	Flicker noise
RSNOISE(Dxx)	Thermal noise due to access resistance
IDNOISE(Dxx)	Shot noise

Table 10-28. Common Plotting and Printing

DC or Transient Analysis	
FLUX(Lxx)	Returns the flux through the inductor Lxx.
ISUB(Xxx.N)	Current flowing into pin N of subcircuit Xxx, where N is the name of the pin in the .SUBCKT statement. The node name can be a node defined in a .global command.
I(dipole_xx[, dipole_yy)	Current difference between two-pin components dipole_xx and dipole_yy of any type (such as resistor, capacitor, source). If dipole_yy and the comma are omitted, the current through dipole_xx is printed. Current is positive when flowing from pin1 to pin2.

Table 10-28. Common Plotting and Printing

I(device.xyz)	Current flowing into pin xyz of any two-pin component type (such as resistor, capacitor, source), where xyz may be POS, NEG, PLUS or MINUS for positive pin, negative pin, plus pin, or minus pin respectively. For example, .PLOT TRAN I(R1.POS) requests a plot of the current going into the positive pin of the resistor R1.
I(OPAxx.xyz)[1]	Current flowing into pin xyz, where xyz may be either POS, NEG, CP1 or CN1 for positive output, negative output, positive control, or negative control respectively.
IX(Xxx.N)	Current flowing into the Nth pin of subcircuit Xxx, where N is the order of the pin in the .SUBCKT statement. that is, IX(XA.1)is the current flowing into the first pin of subcircuit XA. If N is the name of a global node, then Eldo will return the current which enters the X-instance by that global pin.
VX(Xxx.N)	Voltage at the Nth pin of subcircuit Xxx, where N is the order of the pin in the .SUBCKT statement. Works in the same way as IX(Xxx.N) above.
I(dev1[, dev2])	Specifies the current between devices dev1 and dev2. If dev2 and the preceding comma is omitted, ground is assumed.
V(N1[, N2])	Specifies the voltage difference between nodes N1 and N2. If N2 and the preceding comma is omitted, ground is assumed. In the case of subcircuit nodes, N1 has the form (Xnn.N1) where Xnn is the subcircuit instance.
W(WNAME|EXPR)	The value of a waveform WNAME which has been created using the .DEFWAVE command, or the value of the waveform defined by the expression EXPR. .PRINT TRAN W(wave1) or .PRINT TRAN W('V(s)-V(a)')
POW(Xinstance_name)[2]	Returns the power curve dissipated in the Xinstance_name.
POWER	Returns the power dissipated in the entire design. This number is the sum for the power dissipated in R, E, H, I, M, B, D, J elements exclusively.
AC Analysis	
VDB(N1[, N2])	Specifies the magnitude of the voltage difference in dB between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node.
VGD(N1[, N2])	Specifies the group delay (derivative of the phase with respect to the frequency) of the voltage difference between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node. VT may be specified instead of VGD, it is equivalent.

Table 10-28. Common Plotting and Printing

VI(N1[, N2])	Specifies the imaginary part of the voltage difference between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node.
VM(N1[, N2])	Specifies the magnitude of the voltage difference between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node.
VP(N1[, N2])	Specifies the phase of the voltage difference between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node.
VR(N1[, N2])	Specifies the real part of the voltage difference between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node.
IDB(dipole_xx[, dipole_yy])	Specifies the magnitude, in dB, of the current difference between dipole_xx and dipole_yy. If only dipole_xx is specified, the current through dipole_xx is printed.
IGD(dipole_xx[, dipole_yy])	Specifies the group delay (derivative of the phase with respect to the frequency) of the current difference between dipole_xx and dipole_yy. If only dipole_xx is specified, the group delay of the current through dipole_xx is printed. IT may be specified instead of IGD, it is equivalent.
II(dipole_xx[, dipole_yy])	Imaginary part of the current difference between dipole_xx and dipole_yy. If only dipole_xx is specified, the imaginary part of the current through dipole_xx is printed.
IM(dipole_xx[, dipole_yy])	Magnitude of the current difference between dipole_xx and dipole_yy. If only dipole_xx is specified, the magnitude of the current through dipole_xx is printed.
IP(dipole_xx[, dipole_yy])	Phase of the current difference between dipole_xx and dipole_yy. If only dipole_xx is specified, the phase of the current through dipole_xx is printed.
IR(dipole_xx[, dipole_yy])	Real part of the current difference between dipole_xx and dipole_yy. If only dipole_xx is specified, the real part of the current through dipole_xx is printed.
ISUBDB(Xxx.N)	Magnitude, in dB, of the current flowing into pin N of subcircuit Xxx, where N is the name of the pin in the .SUBCKT statement.
ISUBGD(Xxx.N)	Group delay of the current flowing into pin N of subcircuit Xxx, where N is the name of the pin in the .SUBCKT statement.
ISUBI(Xxx.N)	Imaginary part of the current flowing into pin N of subcircuit Xxx where N is the name of the pin in the .SUBCKT statement.

Table 10-28. Common Plotting and Printing

ISUBM(Xxx.N)	Magnitude of the current flowing into pin N of subcircuit Xxx where N is the name of the pin in the .SUBCKT statement.
ISUBP(Xxx.N)	Phase of the current flowing into pin N of subcircuit Xxx where N is the name of the pin in the .SUBCKT statement.
ISUBR(Xxx.N)	Real part of the current flowing into pin N of subcircuit Xxx where N is the name of the pin in the .SUBCKT statement.
IXDB(Xxx.N)	Magnitude in dB, of the current flowing into the Nth pin of subcircuit Xxx, where N is the order of the pin in the .SUBCKT statement. If N is the name of a global node, then Eldo will return the magnitude in dB of the current which enters the X-instance by that global pin.
IXGD(Xxx.N)	Group delay of the current flowing into the Nth pin of subcircuit Xxx, where N is the order of the pin in the .SUBCKT statement. If N is the name of a global node, then Eldo will return the group delay of the current which enters the X-instance by that global pin.
IXI(Xxx.N)	Imaginary part of the current flowing into the Nth pin of subcircuit Xxx. If N is the name of a global node, then Eldo will return the imaginary part of the current which enters the X-instance by that global pin.
IXM(Xxx.N)	Magnitude of the current flowing into the Nth pin of subcircuit Xxx. If N is the name of a global node, then Eldo will return the magnitude of the current which enters the X-instance by that global pin.
IXP(Xxx.N)	Phase of the current flowing into the Nth pin of subcircuit Xxx. If N is the name of a global node, then Eldo will return the phase of the current which enters the X-instance by that global pin.
IXR(Xxx.N)	Real part of the current flowing into the Nth pin of subcircuit Xxx. If N is the name of a global node, then Eldo will return the real part of the current which enters the X-instance by that global pin.
SDB(i, j)	Magnitude in dB, of S parameter Si j.
SR(i, j)	Real part of the S parameter Si j.
SI(i, j)	Imaginary part of the S parameter Si j.
WDB(WNAME)	Magnitude in dB, of the waveform WNAME created using a .DEFWAVE command.
WI(WNAME)	Imaginary part of the waveform WNAME created using a .DEFWAVE command.

Table 10-28. Common Plotting and Printing

WM(WNAME)	Magnitude of the waveform WNAME created using a .DEFWAVE command.
WP(WNAME)	Phase of the waveform WNAME created using a .DEFWAVE command.
WR(WNAME)	Real part of the waveform WNAME created using a .DEFWAVE command.
Noise Analysis	
INOISE[3]	Linear input noise when performing a Noise analysis.
ONOISE	Linear output noise when performing a Noise analysis.
DB(INOISE)	Input noise in dB when performing a Noise analysis.
DB(ONOISE)	Output noise in dB when performing a Noise analysis.
NOISE(compx)	Noise contribution of component compx to the total output noise of the circuit.
NOISE NC[4]	Noise circle for a given value of a Noise Figure (SNF_val). When this value is not specified, the circle is plotted for a Noise Figure value corresponding to the actual circuit

1. In the MOS and bipolar current print commands, the current is positive when it enters the device.
2. In all cases, the power stored in capacitances is ignored, only dissipated power is taken into account. Power is measured only in DC and TRANSIENT analyses.
3. For more information on the use of INOISE and ONOISE, see the example in ".NOISE" on page 744.
4. For more information on NOISE NC see "Two-port Noise Circles" on page 820.

Element Output

`LVnn(Ex)` or `LXnn(Ex)`, where:

`LV` This formulation is used to obtain user-input parameters.

`LX` This formulation is used to obtain computed values such as charges, capacitances, and derivatives.

`nn` Index to select the appropriate output.

`Ex` Name of the element.

Note

In the table below, the `LX` values which correspond to voltage (such as `LX0` for diodes) are computed using the intrinsic nodes, not the extrinsic nodes (the behavior in Eldo releases pre-v5.9).
The names in parentheses are aliases.

Table 10-29. Element Output

Resistors	
LV1	Conductance at analysis temperature
LV2	Resistance at reference temperature
LV3	First temperature coefficient TC1
LV4	Second temperature coefficient TC2
Capacitors	
LV1	Computed capacitance
LX0 (Q)	Charge stored in capacitor
LX1 (I)	Current flowing through capacitor
LX2 (VDIP)	Voltage across capacitor
LX3	Capacitance value
Inductors	
LV1	Computed inductance
LX0	Flux in the inductor
LX1 (VDIP)	Voltage across inductor
LX2 (I)	Current flowing through inductor
LX4	Inductance value
Voltage-Controlled Voltage Sources	
LX0 (VDIP)	Source voltage
LX1 (I)	Current through source
Current-Controlled Current Sources	
LX0 (I)	Current through source
Current-Controlled Voltage Sources	
LX0 (VDIP)	Source voltage
LX1 (I)	Source current
Diodes	
LV1	Diode area factor
LX0 (VDIP)	DC/transient voltage across diode
LX1 (I)	Current through diode; gmin effect is not taken into account

Table 10-29. Element Output

LX2 (GD)	Equivalent conductance
LX3 (Q)	Charge of diode capacitor
LX4	Dynamic current through diode
LX5 (CD)	Total diode capacitance
BJTs	
LV1	Area factor
LV5	FT
LV8	LOG10(IC)
LV9	LOG10(IB)
LV10	BETA
LV11	LOG10(BETA) Current
LX0	VBE
LX1	Base-collector voltage
LX2	Collector current
LX3	Base current
LX4	Gpi = Ib/Vbe, constant vbc
LX5	Gmu = Ib/Vbc, constant vbe
LX6	Gm = Ic/Vbe, constant vbc
LX7	GO = Ic/Vce, constant vbe
LX19 (CPI)	cbe capacitance
LX20 (CMU)	cbc capacitance
JFETs	
LV1	JFET area factor
LX0	VGS
LX1 (VGD)	Gate-drain voltage
LX2 (CGS)	Gate-to-source
LX3 (IDS)	Drain current
LX4 (IGD)	Gate-to-drain current
LX5 (GM)	Transconductance
LX6 (GDS)	Drain-source transconductance
LX9 (QG)	Gate-source charge

Table 10-29. Element Output

LX11 (GD)	Gate-drain capacitance
LX18 (GMB)	Drain-body trans-conductance
MOSFETs	
LV1	Effective channel Length
LV2	Effective channel width
LV3	Effective area of the drain diode
LV4	Effective area of the source diode
LV9 (VT)	Threshold “on” voltage
LV10 (VDSS)	Saturation voltage
LV11	Effective drain diode periphery
LV12	Effective source diode periphery
LV13	Drain resistance (squares)
LV14	Source resistance (squares)
LV15	Charge sharing coefficient
LV16	Effective drain conductance (1/RDeff)
LV17	Effective source conductance (1/RSeff)
LX0 (VBD)	Bulk-drain voltage
LX1 (VBS)	Bulk-source voltage
LX2 (VGS)	Gate-source voltage
LX3 (VDS)	Drain-source voltage
LX4 (IDS)	DC drain current
LX5 (IBS)	DC source-bulk diode current
LX6 (IBD)	DC drain-bulk diode current
LX7 (GM)	DC gate transconductance
LX8 (GDS)	DC drain-source conductance
LX9 (GMBS)	DC substrate transconductance
LX10 (GMIBD)	Conductance of the drain diode
LX11 (GMIBS)	Conductance of the source diode
LX12 (QB)	Bulk charge
LX14 (QG)	Gate charge
LX16 (QD)	Channel charge

Table 10-29. Element Output

LX18 (CGG)	dQg/dVgb
LX19 (CGD)	dQg/dVdb
LX20 (CGS)	dQg/dVsb
LX21 (CBG)	dQb/dVgb
LX22 (CBD)	dQb/dVdb
LX23 (CBS)	dQb/dVsb
LX24 (QBD)	Drain-bulk charge
LX26 (QBS)	Source-bulk charge
LX28	Bulk-source capacitance
LX29	Bulk-drain capacitance
LX32 (CDG)	dQd/dVgb
LX33 (CDD)	dQd/dVdb
LX34 (CDS)	dQd/dVsb

Printing Internal Pin Values

To measure the intrinsic value of a pin you use the same syntax as for measuring its extrinsic value. However, you use an index number greater than the number of pins available that corresponds to the pins position. For example, the source voltage pin is the third pin of a MOSFET and there are 4 pins in total. Therefore, to obtain the intrinsic source voltage of this MOSFET you would use:

```
.PLOT V(Mx.7)
```

To obtain the intrinsic value of the drain current pin of a JFET you would use:

```
.PLOT I(Jx.4)
```

Note

If the position number used is greater than the index available for the device it will return a value of zero.

Monitoring of Hierarchical Nodes

More information related to the internal status of devices can be displayed. The following is a list of variables displayed together with the syntax to address them. This output format is used to specify nodes that lie within subcircuits. The node itself cannot be referenced directly from the 'top-level' of the circuit and so it must be addressed through the levels of the subcircuit by separating elements with periods (.) in the output control statement. The example below illustrates this output format.

```
.subckt sc1 n10 n12
r10 n10 n11 0.2
x2 n11 n12 sc2
.ends sc1
.subckt sc2 n20 n22
r20 n20 n21 0.1k
r22 n21 n22 0.1k
.ends sc2
x1 a b sc1
.print tran v(x1.x2.n21) v(a) v(b)
```

The above example specifies the output of three nodes. The node `x1.x2.n21` is created as the second pin of resistor `r20`, which is an element of the subcircuit `sc2`, instantiated using instance `x2`. The instance `x2` is itself nested in the subcircuit `sc1`, instantiated using instance `x1`.

Transmission Lines

- **I(**Txx.N1**) I(**Txx.N2**) I(**Txx.N3**) I(**Txx.N4**)**
  Prints the current out of transmission lines.

Two-port Parameters

Table 10-30. Two-port Parameters

Two-port Stability Factors (for more information, see Two-Port Stability Factors of the *Eldo RF User's Manual*)	
KFACTOR	Computes the stability factor for 2-ports. $\frac{(1.0 - \lvert S11\rvert^2 - \lvert S22\rvert^2 - \lvert S11 \times S22 - S12 \times S21\rvert^2)}{(2 \times \lvert S12 \times S21\rvert)}$ S11, S22, and so on, are the S parameters.
BFACTOR	Rollet stability factor. $BFACTOR = \frac{1 - \lvert S_{22}\rvert^2}{\lvert S_{11} - \Delta S_{22}{*}\rvert + \lvert S_{22} \cdot S_{11}\rvert}$
MUFACTOR	Rollet stability factor. $MUFACTOR = \frac{1 + \lvert S_{11}\rvert^2 - \lvert S_{22}\rvert^2 + \lvert\Delta\rvert^2}{\lvert S_{11} - \Delta S_{22}{*}\rvert}$ where $\Delta = S_{11} \cdot S_{22} - (S_{12} \cdot S_{21})$
MUFACTOR_L	Rollet stability factor (Load). $\mathrm{MUFACTOR\_L} = \frac{1 - \lvert S_{11}\rvert^2}{\lvert S_{22} - (S_{11})^* \cdot \Delta\rvert + \lvert S_{12} \cdot S_{21}\rvert}$ where Δ is as above.

Table 10-30. Two-port Parameters

MUFACTOR_S	Rollet stability factor (Source). $$\text{MUFACTOR\_S} = \frac{1 - \lvert S_{22} \rvert^2}{\lvert S_{11} - (S_{22})^* \cdot \Delta \rvert + \lvert S_{21} \cdot S_{12} \rvert}$$ where Δ is as above.
Two-port Noise Parameters (for more information, see Two-Port Noise Parameters of the *Eldo RF User's Manual*)	
NFMIN_mag	Minimal noise figure (magnitude) of the two-port. $$NFMIN = 1 + 2 \cdot \frac{Re\{C_{12}^A\} + GOPT \cdot C_{11}^A}{4 \cdot k \cdot T}$$
NFMIN_dB	Minimal noise figure of the two-port (in decibels).
GOPT	Real part of the optimal source admittance. $$GOPT = \frac{1}{C_{11}^A}\sqrt{C_{11}^A \cdot C_{22}^A - (Im\{C_{12}^A\})^2}$$
BOPT	Imaginary part of the optimal source admittance. The sign convention of this parameter can be changed using "NOISE_SGNCONV" on page 994. $$BOPT = \frac{Im\{C_{12}^A\}}{C_{11}^A}$$
RNEQ	Equivalent noise resistance. $$RNEQ = \frac{C_{11}^A}{4 \cdot k \cdot T}$$
YOPT	Source admittance that produces the minimal noise figure. $YOPT = GOPT + j \cdot BOPT$
GAMMA_OPT	Complex quantity of the optimal reflection coefficient associated with the minimum noise figure. $\text{GAMMA\_OPT} = \text{GAMMA\_OPT\_MAG} \cdot \exp(j \cdot \text{PHI\_OPT})$
GAMMA_OPT_MAG	Magnitude of the optimal reflection coefficient associated with the minimum noise figure. $\text{GAMMA\_OPT\_MAG} = \lvert \Gamma_{opt} \rvert$
PHI_OPT	Angle of the optimal reflection coefficient associated with the minimum noise figure. The sign convention of this parameter can be changed using "NOISE_SGNCONV" on page 994. $\text{PHI\_OPT} = \angle\Gamma_{opt}$

Table 10-30. Two-port Parameters

Two-port Noise Circles (for more information, see Two-Port Noise Circles of the *Eldo RF User's Manual*)	
NC	A noise circle is plotted for a given value of a Noise Figure (SNF_val). When this value is not specified, the circle is plotted for a Noise Figure value corresponding to the actual circuit. $\text{center} = \frac{\Gamma_{opt}}{1 + N_i}$ $\text{radius} = \sqrt{\frac{N_i^2 + N_i \cdot (1 - \lvert\Gamma_{opt}\rvert^2)}{1 + N_i}}$ $N_i = \frac{(\text{SNF\_val} - NFMIN) \cdot \lvert 1 + \Gamma_{opt}\rvert^2}{4 \cdot \text{RNEQ}}$
Two-port Constant Gain Circles (for more information, see Two-Port Constant Gain Circles of the *Eldo RF User's Manual*)	
GAC	Available Gain Circle. Determines constant gain contour at the input port. With respect to the definition of `GA` and `GP`, `GAC` represents an optimum match at the output port. $\text{center} = \frac{GA(S_{11}^* - S_{22}\Delta^*)}{\lvert S_{21}\rvert^2 + GA(\lvert S_{11}\rvert^2 - \lvert\Delta\rvert^2)}$ where $\Delta = (S_{11}S_{22} - S_{21}S_{12})$ $\text{radius} = \frac{\sqrt{1 - 2KFACTOR\lvert S_{21}S_{12}\rvert\frac{GA}{\lvert S_{21}\rvert^2} + \left(\lvert S_{21}S_{12}\rvert\frac{GA}{\lvert S_{21}\rvert^2}\right)^2}}{\left\lvert 1 + \frac{GA}{\lvert S_{21}\rvert^2}(\lvert S_{11}\rvert^2 - \lvert\Delta\rvert^2)\right\rvert}$

Table 10-30. Two-port Parameters

GPC	Power Gain Circle. Determines constant gain contour at the output port. With respect to the definition of **GA** and **GP**, **GPC** represents an optimum match at the input port. $$\text{center} = \frac{GP(S_{22}^* - S_{11}\Delta^*)}{\lvert S_{21}\rvert^2 + GP(\lvert S_{22}\rvert^2 - \lvert\Delta\rvert^2)}$$ $$\text{radius} = \frac{\sqrt{1 - 2KFACTOR\lvert S_{21}S_{12}\rvert\frac{GP}{\lvert S_{21}\rvert^2} + \left(\lvert S_{21}S_{12}\rvert\frac{GP}{\lvert S_{21}\rvert^2}\right)^2}}{\left\lvert 1 + \frac{GP}{\lvert S_{21}\rvert^2}(\lvert S_{22}\rvert^2 - \lvert\Delta\rvert^2)\right\rvert}$$
Two-port Gain Parameters (for more information, see Two-Port Gain Parameters of the *Eldo RF User's Manual*)	
GA_mag	Available power Gain (magnitude). This is the ratio of the power available from the two-port circuit to the power available from the source when the load is conjugately matched to the output port of the circuit (Γ_L = conj(Γ_{OUT})). $$GA = \frac{1 - \lvert\Gamma_s\rvert^2}{\lvert 1 - S_{11}\Gamma_s\rvert^2}\lvert S_{21}\rvert^2\frac{1}{1 - \lvert\Gamma_{OUT}\rvert^2}$$
GA_dB	Available power Gain (in decibels).
GAM_mag	Maximum Available power Gain (magnitude). When the input port of the circuit is conjugately matched to the source impedance and the output port of the circuit is conjugately matched to the load impedance (Γ_S = conj(Γ_{IN}) and Γ_L = conj(Γ_{OUT})). For a bilateral case: $GAM = \left\lvert\frac{S_{21}}{S_{12}}\right\rvert(KFACTOR - \sqrt{KFACTOR^2 - 1})$ if $KFACTOR > 1$ $GAM = \left\lvert\frac{S_{21}}{S_{12}}\right\rvert$ if $KFACTOR \le 1$ For the **KFACTOR**, see "KFACTOR" on page 818. For a unilateral case, see "GAUM_mag" on page 822.
GAM_dB	Maximum Available power Gain (in decibels).
GASM_mag	Maximum Available Stable Gain (magnitude). See the **GAM** definition above. $$GASM = \left\lvert\frac{S_{21}}{S_{12}}\right\rvert$$
GASM_dB	Maximum Available Stable Gain (in decibels).

Table 10-30. Two-port Parameters

GP_mag	Power Gain (magnitude). This is the ratio of the power delivered to the load to the power input to the two-port circuit when the input port of the circuit is conjugately matched to the source impedance (Γ_S = conj(Γ_{IN})). $$GP = \frac{1-\lvert\Gamma_L\rvert^2}{\lvert 1-S_{22}\Gamma_L\rvert^2}\lvert S_{21}\rvert\frac{1}{1-\lvert\Gamma_{IN}\rvert^2}$$ $$\Gamma_L = \frac{Z_L - Z_0}{Z_L + Z_0}$$ $$\Gamma_{IN} = S_{11} + \frac{S_{12}S_{21}\Gamma_L}{1-S_{22}\Gamma_L}$$ Z_S Source impedance Z_L Load impedance Γ_S Source reflection coefficient Γ_L Load reflection coefficient Z_0 Characteristic impedance (by default 50 Ω; to modify this value see “ZCHAR=VAL” on page 1022).
GP_dB	Power Gain (in decibels).
GAUM_mag	Maximum Unilateral transducer power Gain (magnitude). This is the transducer gain when the circuit ports are both optimally matched (Γ_S = conj(Γ_{IN}) and Γ_L = conj(Γ_{OUT}) and S_{12} = 0). $$GAUM = \frac{1}{1-\lvert S_{11}\rvert^2}\lvert S_{21}\rvert^2\frac{1}{1-\lvert S_{22}\rvert^2}$$
GAUM_dB	Maximum Unilateral transducer power Gain (in decibels).
TGP_mag	Transducer Power Gain (magnitude). This is the ratio of the power delivered to the load to the power available from the source. $$TGP = \frac{1-\lvert\Gamma_s\rvert^2}{\lvert 1-S_{11}\Gamma_s\rvert^2}\lvert S_{21}\rvert\frac{1-\lvert\Gamma_L\rvert^2}{\lvert 1-\Gamma_{OUT}\Gamma_L\rvert^2}$$
TGP_dB	Transducer Power Gain (in decibels).
Two-port Stability Circles (for more information, see Two-Port Stability Circles of the *Eldo RF User’s Manual*)	
SSC	Source Stability Circle. It determines the locus of Γ_S which produce $\lvert\Gamma_{OUT}\rvert = 1$. $$\text{center} = \frac{S_{22}\Delta^* - S_{11}^*}{\lvert\Delta\rvert^2 - \lvert S_{11}\rvert^2}$$ $$\text{radius} = \left\lvert\frac{S_{12}S_{21}}{\lvert\Delta\rvert^2 - \lvert S_{11}\rvert^2}\right\rvert$$

Table 10-30. Two-port Parameters

LSC	Load Stability Circle. It determines the locus of Γ_L which produce $\|\Gamma_{IN}\| = 1$. $\text{center} = \dfrac{S_{11}\Delta^* - S_{22}^*}{\|\Delta\|^2 - \|S_{11}\|^2}$ $\text{radius} = \left\| \dfrac{S_{12}S_{21}}{\|\Delta\|^2 - \|S_{22}\|^2} \right\|$

Plotting an analog signal as a digital bus

To enable the plotting of an analog signal as a digital bus, a setup command is required (**.DEFPLOTDIG**) to be used in conjunction with the **VDIG** parameter of the **.PLOT** command.

```
.DEFPLOTDIG VTH1=2.2 VTH2=2.7
.PLOT TRAN VDIG(n2)
```

When **.DEFPLOTDIG** is used in conjunction with **.PLOT VDIG**, the signal node n2 is plotted as a digital curve and will only have values of “1” or “0”.

- **.DEFPLOTDIG** [**VTH**[**1**]=VAL [**VTH2**=VAL]]

  This is used as a precursor to **VDIG** in order to plot an analog curve as digital with reference to any stated threshold voltage(s).

- **VTH**[**1**]=VAL

  If a voltage threshold is specified, the bus of an analog signal is plotted as a bus (hexadecimal format), else all the different signals of the bus are plotted separately in the waveform viewer as analog waves. (VTH and VTH1 are equivalent to ensure backwards compatibility.)

- **VTH2**=VAL

  This can be used to plot the indeterminate value as shown below:

  When only **VTH1** is given:
  If value < **VTH** then logic state 0.
  If value > **VTH** then logic state 1.

  When both **VTH1** and **VTH2** are given:
  If value < **VTH1** then logic state 0.
  If **VTH1** < value < **VTH2** then state X.
  If value > **VTH2** then logic state 1.

Examples

```
.PLOT TRAN V(a) V(b)
```

Specifies that the voltages for **V**(a) and **V**(b) be plotted. EZwave will plot both **V**(a) and **V**(b) on the Y-axis with the time plotted on the X-axis.

```
.PLOT TRAN V(a) (VERSUS) V(b)
```

Specifies that EZwave will plot **V**(a) on the Y-axis against **V**(b), which will be plotted on the X-axis.

```
.plot tran v(1) v(2) v(3, 4) i(v1, v2)
```

Specifies that the voltage at the nodes 1 and 2, the voltage difference between the nodes 3 and 4, and the current difference of the voltage sources v1 and v2 be plotted.

```
.plot ac vdb(4)(–50, 50) idb(v6)(–75, 75)
```

This is the same example as the last except that the dB plot limits have changed. The new limits are –50dB to +50dB for the voltage at node 4, and –75dB to +75dB for the current through voltage source v6.

```
.plot ac vm(4) unit=Watts
```

Specifies that the voltage at node 4 be plotted, and that the Y-axis units of the plot be Watts.

```
.PLOT NOISE NOISE(r1)
```

Eldo will plot the noise generated by the resistor r1 in the circuit.

```
.plot noise noise(X1)
```

Returns the total noise contribution of a subcircuit instance (X1). Its value is the sum of the noise of all elements that are part of the specified subcircuit instance.

```
.FOUR LABEL = t1 V(a)
.PLOT FOUR FOURDB(t1)
```

Eldo will plot the value (in dB) at 5 Meg for the FFT done on V(a).

```
.PLOT VX(M1.7)
.PLOT VX(Q1.3)
.PLOT VX(R1.2)
```

Eldo will plot values according to the following: VX(M1.7) refers to the intrinsic source voltage, VX(Q1.3) refers to the extrinsic emitter voltage, VX(R1.2) refers to the ‘second’ pin of resistor R1.

```
.PLOT TRAN w1=v(X1.X2.X3.X4.OUT)
```

Here, the wave v(X1.X2.X3.X4.OUT) will be plotted inside EZwave, but the legend will be w1 and not v(X1.X2.X3.X4.OUT).

```
.PLOT AC S(1,1) S(2,2) (SMITH)
```

Shows syntax for specifying EZwave to display the waves in a Smith chart. In the example above, both S(1,1) and S(2,2) will be on the same Smith chart.

```
.PLOT AC S12 (POLAR)
```

Shows syntax for specifying EZwave to display the waves in a Polar chart.

```
.PLOT NOISE NFMIN_MAG
```

Plots the minimal noise figure for two port noise.

```
.PLOT NOISE YVAL(RNEQ, 10k)
```

In the example above, the y-axis value of waveform `WAVE` is returned when the equivalent noise resistance reaches 10k.

```
.PLOT OPT WOPT(FOO)
```

Plots the extracted wave `FOO` which was generated during optimization.

```
.DSP LABEL=LBL2 MODEL=PSD_PERIODO V(1)
.PLOT DSP DSPDB(LBL2)
```

Displays DSP results. Eldo will plot the values (in dB) for the DSP done on `V(1)`.

```
.DATA dataname component1 component2
+ x1 c1y1 c2y1
+ x2 c1y2 c2y1
...
+ xn c1yn c2yn
.ENDDATA

.PLOT TRAN DATA(dataname,component2)
```

Eldo will plot a wave defined with the **`.DATA`** statement.

```
.PLOT FOUR fourdb(foo) (CONTINUOUS)
```

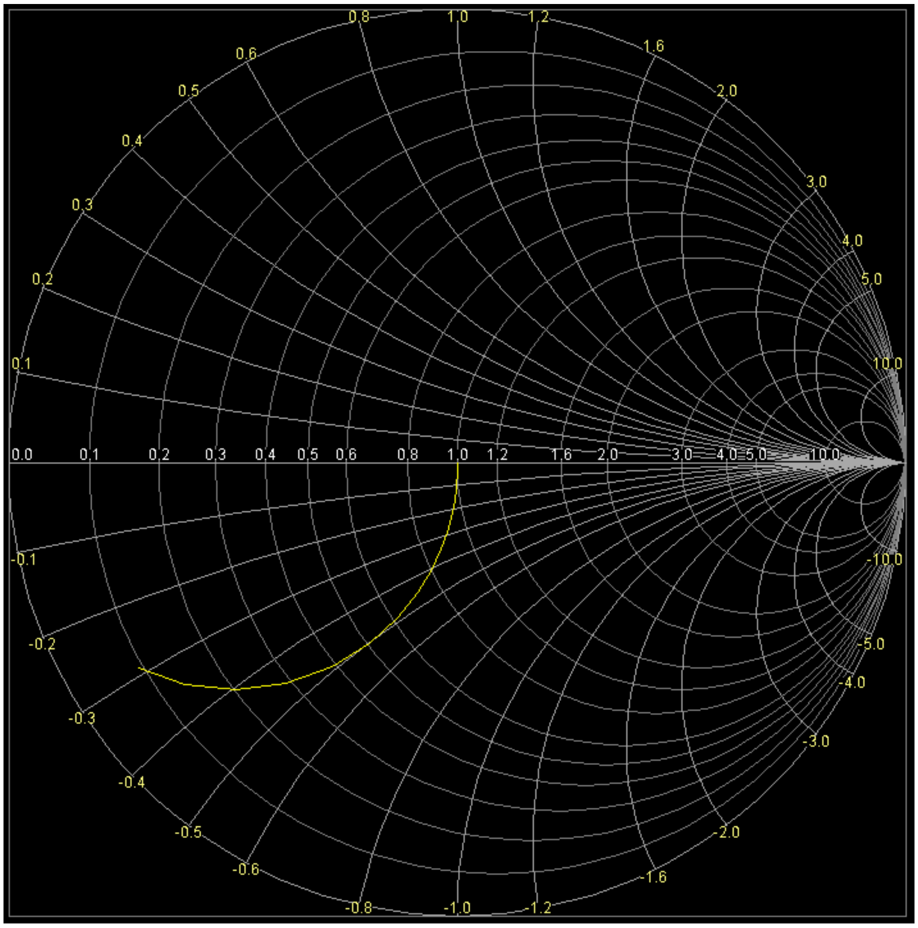
Plots the FFT waveforms in continuous drawing mode instead of the default spectral mode.

The following full netlist example shows how to plot the S11, Zin and Yin of a node (IN) on a Smith chart using STOSMITH, ZTOSMITH, and YTOSMITH functions.

```
VIN IN 0 RPORT=50 IPORT=1 FOUR FUND1 MA (1) 1 0
RIN IN 1 RIN
CIN IN 0 CIN
LIN 1 0 LIN
.PARAM F1=1G Z0=50 RIN=Z0
.PARAM CIN=1p LIN=1f
.SST FUND1=F1 NHARM1=1
.STEP PARAM F1 DEC 10 1k 10G
.DEFWAVE ZIN=(1+S(1,1))/(1-S(1,1))*50
.DEFWAVE YIN=1/W(ZIN)

.EXTRACT FSST LABEL=S_EXT STOSMITH(YVAL(S(1,1),F1))
.PLOT MEAS(S_EXT) (SMITH,50)
.EXTRACT FSST LABEL=Z_EXT ZTOSMITH(YVAL(W(ZIN),F1),50)
.PLOT MEAS(Z_EXT) (SMITH,50)
.EXTRACT FSST LABEL=Y_EXT YTOSMITH(YVAL(W(YIN),F1),50)
.PLOT MEAS(Y_EXT) (SMITH,50)

.END
```

As can be seen in EZwave, the three functions produce the same waveform.

Figure 10-5. Smith chart plot using STOSMITH/ZTOSMITH/YTOSMITH functions



.PLOTBUS

Plotting of Bus Signals

```
.PLOTBUS BNAME [VTH[1]=VAL [VTH2=VAL]]
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP] [(ANALOG)]
.PLOTBUS BNAME[MSB:LSB]|BNAME<MSB:LSB>|BNAMEMSB:LSB
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP] [(ANALOG)]
```

This command plots all the bits of a bus, previously defined via the **.SETBUS** and using **.SIGBUS** to send a value. **.SETBUS** can be implicitly declared, providing that `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` or `BNAMEMSB:LSB` syntax is used.

Parameters

- `BNAME`

  Bus name, previously defined via the **.SETBUS** command, unless `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` or `BNAMEMSB:LSB` syntax is used, in which case **.SETBUS** is implicitly declared.

- `VTH[1]=VAL`

  If a voltage threshold is specified, the bus of an analog signal is plotted as a bus (hexadecimal format), else all the different signals of the bus are plotted separately in the waveform viewer as analog waves. (`VTH` and `VTH1` are equivalent to ensure backward compatibility.)

- `VTH2=VAL`

  This can be used to plot the indeterminate value as shown below:

  When only **VTH1** is given:
  If value < **VTH** then logic state 0.
  If value > **VTH** then logic state 1.

  When both **VTH1** and **VTH2** are given:
  If value < **VTH1** then logic state 0.
  If **VTH1** < value < **VTH2** then state X.
  If value > **VTH2** then logic state 1.

- `MSB:LSB`

  Series of bit names, the most significant bit being defined first. This mechanism can be used to implicitly declare **.SETBUS**.

- **BASE**

  Keyword indicating that the bus signal number system is to be defined.

  OCT

  Keyword indicating that bus signals are defined in octal.

  DEC

  Keyword indicating that bus signals are defined in decimal. Default.

BIN

Keyword indicating that bus signals are defined in binary.

HEX

Keyword indicating that bus signals are defined in hexadecimal.

- **RADIX**=**UNSIGNED**|**2COMP**

  Defines how negative values inside bus patterns are managed.

  UNSIGNED—negative values are forbidden (default)

  2COMP—the two's complement of the value is done

- **ANALOG**

  Allows to plot the bus as an analog waveform using the bus decimal values. Unit is Volts.

Examples

In order to display the output of a bus, you can use the following syntax:

```
.PLOTBUS Y<instance_name> -> <vector_name>
```

The following example shows how a second voltage threshold can be used:

```
.plotbus foo vth=1 vth2=4
.checkbus OUT_BUS_2 vth=1 vth2=4 base=bin
+ 1000ps 01
+ 7000ps x
+ 12000ps 11
+ 15000ps 01
+ 19500ps 01
+ 20000ps 10
+ 22000ps 10
+ 30000ps 01
```

Refer to ".CHECKBUS" on page 561 for more information.

.PLOTBUS can be used to implicitly declare **.SETBUS**, providing that BNAME[MSB:LSB] or BNAME<MSB:LSB> or BNAMEMSB:LSB syntax is used as shown below:

```
.PLOTBUS SELECT[2:0]
.PLOTBUS DOUT<3:0> VTH=1.65
```

is equivalent to:

```
.SETBUS SELECT SELECT[2] SELECT[1] SELECT[0]
.PLOTBUS SELECT
.SETBUS DOUT DOUT<3> DOUT<2> DOUT<1> DOUT<0>
.PLOTBUS DOUT VTH=1.65
```

The following example shows the most significant bit as 1 and the least significant as 5, where the bus name is `foo`.

```
.PLOTBUS foo1:5 vth=3
```

.PRINT

Printing of Results

```
.PRINT [ANALYSIS] [alias_name=]OVN
```

The **`.PRINT`** command defines the contents of a tabular listing of any number of output variables. The sampling period of the **`.PRINT`** command is equivalent to the **`TPRINT`** parameter of the .TRAN command.

By default, the quantities listed in a **`.PRINT`** command are not written to the ASCII *.chi* output file. This avoids creating huge *.chi* output files in the case of large simulations. The ASCII outputs are still supported for SPICE compatibility reasons, although seldom used. To force Eldo to generate these output quantities in the ASCII output file, specify the `-ascii` command-line flag when invoking Eldo or include the option **`ASCII`** in the netlist (see "`ASCII`" on page 998). Symbols such as *, +, and so on, will be used to identify each quantity, to create ASCII *waveforms* in the *.chi* output file.

Parameters

- `ANALYSIS`

  Optional. Specifies the analysis type, which can be useful in the case of multiple types of analysis in the *.cir* file. Can be one of the following:

  `DC`
  Specifies that the plots are required for a DC analysis. Note, not available for a single analysis.

  `AC`
  Specifies that the plots are required for an AC analysis.

  `TRAN`
  Specifies that the plots are required for a transient analysis.

  `NOISE`
  Specifies that the plots are required for a noise analysis.

  `SSTAC|SSTXF|SSTNOISE|SSTJITTER|TMODSST|FMODSST|FOURMODSST|TSST|FSST|SSTSTABIL`
  Please refer to the Display Command Syntax chapter of the *Eldo RF User's Manual* for more information regarding these RF options.

  `FOUR`
  Displays FFT results. See "`FOURxx(label_name)`" on page 799 for display options.

  `DSP`
  Displays DSP results. See "`DSPxx(label_name)`" on page 799 for display options.

- `alias_name`

  Refers to the wave name in the ASCII and binary output files. The `alias_name` will be the legend displayed inside the wave viewer for the plotted wave as specified by the plot specifications in `OVN`.

- `OVN`

  Output variable name. A list of plot specifications can follow. Mandatory. These are listed in "Plot Specifications (OVN)" on page 799 of the **`.PLOT`** command.

Related options

`NOASCII` (see "NOASCII" on page 1004)

Examples

```
.print ac vdb(n2, n4) vp(n2, n4)
```

Requests that the output of the magnitude between node `n2` and node `n4` in dB and the phase between `n2` and `n4` in degrees be printed.

```
.print noise inoise onoise noise(m1) noise(m2)
```

Requests that the input and output noise from a Noise analysis be printed, together with the noise contributions of components **`m1`** and **`m2`** to the total output noise.

```
.defwave z=(V(a)–V(b))/i(r1)
.print ac wr(z) wi(z)
```

Requests print out of the real and imaginary parts of the new waveform, `z`, created by the **`.DEFWAVE`** command.

.PRINTBUS

Printing of Bus Signals

```
.PRINTBUS BNAME [VTH[1]=VAL [VTH2=VAL]]
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP]
.PRINTBUS BNAME[MSB:LSB]|BNAME<MSB:LSB>|BNAMEMSB:LSB
+ [BASE=OCT|DEC|BIN|HEX] [RADIX=UNSIGNED|2COMP]
```

This command prints all the bits of a bus, previously defined via the **.SETBUS** and using **.SIGBUS** to send a value. **.SETBUS** can be implicitly declared, providing that `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` or `BNAMEMSB:LSB` syntax is used.

Parameters

- `BNAME`

  Bus name, previously defined via the **.SETBUS** command, unless `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` or `BNAMEMSB:LSB` syntax is used, in which case **.SETBUS** is implicitly declared.

- `VTH[1]=VAL`

  If a voltage threshold is specified, the bus of an analog signal is plotted as a bus (hexadecimal format), else all the different signals of the bus are plotted separately in the waveform viewer as analog waves. (`VTH` and `VTH1` are equivalent to ensure backwards compatibility.)

- `VTH2=VAL`

  This can be used to plot the indeterminate value as shown below:

  When only **VTH1** is given:
  If value < **VTH** then logic state 0.
  If value > **VTH** then logic state 1.

  When both **VTH1** and **VTH2** are given:
  If value < **VTH1** then logic state 0.
  If **VTH1** < value < **VTH2** then state X.
  If value > **VTH2** then logic state 1.

- `MSB:LSB`

  Series of bit names, the most significant bit being defined first. This mechanism can be used to implicitly declare **.SETBUS**.

- **BASE**

  Keyword indicating that the bus signal number system is to be defined.

  OCT

  Keyword indicating that bus signals are defined in octal.

  DEC

  Keyword indicating that bus signals are defined in decimal. Default.

BIN

Keyword indicating that bus signals are defined in binary.

HEX

Keyword indicating that bus signals are defined in hexadecimal.

- **RADIX**=**UNSIGNED**|**2COMP**

  Defines how negative values inside bus patterns are managed.

  UNSIGNED—negative values are forbidden (default)

  2COMP—the two's complement of the value is done

Examples

In order to display the output of a bus, you can use the following syntax:

```
.PRINTBUS Y<instance_name> -> <vector_name>
```

The following example shows how a second voltage threshold can be used:

```
.printbus foo vth=1 vth2=4
.checkbus OUT_BUS_2 vth=1 vth2=4 base=bin
+ 1000ps 01
+ 7000ps x
+ 12000ps 11
+ 15000ps 01
+ 19500ps 01
+ 20000ps 10
+ 22000ps 10
+ 30000ps 01
```

Refer to ".CHECKBUS" on page 561 for more information.

.PRINTBUS can be used to implicitly declare **.SETBUS**, providing that `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` or `BNAMEMSB:LSB` syntax is used as shown below:

```
.PRINTBUS SELECT[2:0]
.PRINTBUS DOUT<3:0> VTH=1.65
```

is equivalent to:

```
.SETBUS SELECT SELECT[2] SELECT[1] SELECT[0]
.PRINTBUS SELECT
.SETBUS DOUT DOUT<3> DOUT<2> DOUT<1> DOUT<0>
.PRINTBUS DOUT VTH=1.65
```

The following example shows the most significant bit as 1 and the least significant as 5, where the bus name is `foo`.

```
.PRINTBUS foo1:5 vth=3
```

Example *.chi* file extract for **.PRINTBUS** `FILTWD_IN` **BASE=BIN**:

```
1: BUS(FILTWD_IN)
TIME        1
X

0.0         0001
1.0000E-07  0001
2.0000E-07  0001
3.0000E-07  0001
4.0000E-07  0001
5.0000E-07  0001
6.0000E-07  0001
7.0000E-07  0001
8.0000E-07  0001
9.0000E-07  0001
1.0000E-06  0000
1.1000E-06  0010
1.2000E-06  0010
1.3000E-06  0010
1.4000E-06  0010
[...]
```

.PRINTFILE

Print Tabular Output File

```
.PRINTFILE [ANALYSIS] OVN FILE=filename [STEP=value]
+ [START=value] [STOP=value] [FORMAT=DATA]
```

This command defines the contents of a tabular listing of any number of output variables, and dumps them to the specified file in ASCII format or as a **.DATA** statement. Multiple **.PRINTFILE** commands can share an output file, with variables being added one after the other, or multiple files can be defined.

The number of decimal places in the output of **.PRINTFILE** changed between Eldo v6.7 and v6.8 from six to five. It became linked with option **NUMDGT**, which has a default value of 5. For backward compatibility, set option **NUMDGT** to 6.

Parameters

- ANALYSIS

  Optional. Specifies the analysis type, which can be useful in the case of multiple types of analysis in the *.cir* file. Can be one of the following:

  DC|DCSWEEP
  Specifies that the plots are required for a DC analysis.

  AC
  Specifies that the plots are required for an AC analysis.

  TRAN
  Specifies that the plots are required for a transient analysis.

  NOISE
  Specifies that the plots are required for a noise analysis.

  SSTAC|SSTXF|SSTNOISE|SSTJITTER|TMODSST|FMODSST|FOURMODSST|TSST|FSST|SSTSTABIL
  Please refer to the Display Command Syntax chapter of the *Eldo RF User's Manual* for more information regarding these RF options.

  FOUR
  Outputs FFT results. See "FOURxx(label_name)" on page 799 for display options.

  DSP
  Outputs DSP results. See "DSPxx(label_name)" on page 799 for display options.

  SWEEP
  Specifies that the plots are required during a multiple-run simulation if defined.

- OVN

  Output variable name. The syntax for specifying the list of plot specifications to be monitored is provided in "Plot Specifications (OVN)" on page 799. Mandatory.

- `FILE=filename`

  Specifies the file name which the table will be printed.

- `STEP=value`

  Performs a sampling of the variable(s). If this parameter is specified in more than one **.PRINTFILE** command using the same output file then the smallest value for `STEP` is used. It can only be specified for databases that support multiple x-axes in the same simulation, for example JWDB. Other databases will ignore this parameter.

- `START=value`

  Defines the first x-axis value that must be printed in the output file.

- `STOP=value`

  Defines the last x-axis value that must be printed in the output file.

- **FORMAT**=**DATA**

  When specified, the output of **.PRINTFILE** will be saved as a **.DATA** statement.

Related options

NUMDGT (see "NUMDGT=INTEGER_VAL" on page 996)

Examples

```
.printfile tran v(1) i(r1) file="output.txt"
```

This example requests that the voltage on node 1 and the current through r1 be printed to the file *output.txt*.

```
.step param vdd 1 5 1
.step param vbb 2 6 2
.extract dc label=p1 pval(vdd)
.extract dc label=p2 pval(vbb)
.extract dc label=m1 i(r1)
.printfile sweep meas(p1) meas(p2) meas(M1) file=foo.txt format=data
```

This example specifies the output of **.PRINTFILE** will be saved as a **.DATA** statement. File *foo.txt* will contain:

```
.DATA dataname MEAS(P1) MEAS(P2) MEAS(M1)
+   1.000000e+00 2.000000e+00 5.000000e-01
+   1.000000e+00 4.000000e+00 2.500000e-01
+   1.000000e+00 6.000000e+00 1.666667e-01
+   2.000000e+00 2.000000e+00 1.000000e+00
+   2.000000e+00 4.000000e+00 5.000000e-01
+   2.000000e+00 6.000000e+00 3.333333e-01
+   3.000000e+00 2.000000e+00 1.500000e+00
+   3.000000e+00 4.000000e+00 7.500000e-01
+   3.000000e+00 6.000000e+00 5.000000e-01
+   4.000000e+00 2.000000e+00 2.000000e+00
+   4.000000e+00 4.000000e+00 1.000000e+00
+   4.000000e+00 6.000000e+00 6.666667e-01
+   5.000000e+00 2.000000e+00 2.500000e+00
+   5.000000e+00 4.000000e+00 1.250000e+00
```

```
+  5.000000e+00 6.000000e+00 8.333333e-01
.ENDDATA
```

Below is another example where the output of **.PRINTFILE** will be saved as a **.DATA** statement.

```
.tran 1n 10n
.defwave tran foo=xaxis
.printfile tran w(foo) I(r1) file=foo2.txt format=data
```

File *foo2.txt* will contain:

```
.DATA dataname W(FOO) I(R1)
+  0.000000e+00 0.000000e+00
+  5.000000e-12 1.500000e-03
+  1.500000e-11 4.500000e-03
+  3.500000e-11 1.050000e-02
+  7.500000e-11 2.250000e-02
+  1.550000e-10 4.650000e-02
+  3.150000e-10 9.450000e-02
+  6.350000e-10 1.905000e-01
+  1.275000e-09 3.825000e-01
+  2.555000e-09 7.665000e-01
+  5.115000e-09 1.534500e+00
+  1.000000e-08 3.000000e+00
.ENDDATA
```

.PROBE

Output Shortform

```
.PROBE [ANALYSIS] [ALL|I|IX|ISUB|PORT|PRINT|Q|S|SG|SPARAM|V|VN|VTOP|
+ VX|VXN|W|WTOP] [MASK=mask_name] [EXCEPT=pattern] [PRINT] [STEP=val]
+ [DELTA=val]
.PROBE [ANALYSIS] [MASK=mask_name] [EXCEPT=pattern] [alias_name=] OVN
+ [PRINT] [STEP=val] [DELTA=val]
```

At first glance, the difference in using the **.PROBE** syntax above instead of using the **.PRINT** command may not be clear. When using **.PRINT**, simulation results are written to the *.chi* log file in ASCII format. When using **.PROBE**, the set of signals to be monitored is specified in the same way, but results are written to binary output files (*.wdb*). The *.wdb* format is the binary format used by EZwave. Thus, the simulated results are available for post-processing. Other advantages are binary storage saves significant disk space and is faster to read/write. **.PROBE** and **.PLOT** both dump waves to a database but **.PLOT** also adds some information for "plot combinations," which means the ability to group waves into graphs, use Smith charts, change wave representation (scattered, histogram, and so on).

Many different simulation results can be created by Eldo. The simulator can output simple node voltages, but also currents through devices, currents through device or subcircuit pins, power quantities, S parameters, internal variables from device models, and so on. All these results are direct *raw* results from a simulation. The following information, together with the tables shown in "Plot Specifications (OVN)" on page 799, indicate the exact syntax to use for each category. For example:

```
.PROBE TRAN V(OUT)
.PROBE DC ISUB(XBIAS.VOUT)
```

The **.PROBE** command, without specified parameters, forces Eldo to save all node voltages in the binary output file. The **.PROBE** command is a short way of specifying that all nodes should be output. It is not possible to mix analysis types in one command line syntax, therefore the following statement will be rejected:

```
.probe ac I dc V tran I V S
```

It is also not possible to mix in one command line syntax general probe commands (such as **.PROBE v**, **.PROBE I**, **.PROBE vtop**) with specific probe commands (such as **.PROBE v(a)**). Two separate commands must be specified.

When the circuit has more than 1000 nodes, **.PROBE** is ignored unless the **LIMPROBE** parameter, which specifies the maximum number of nodes which may be probed, is increased using the **.OPTION** command.

Note

The saving of all or large numbers of nodes can generate very large output files.

In order to analyze simulation results for huge circuits, and in case the user is only interested in displaying part of the circuit, a **.PROBE** command with nodes defined via both hierarchy and wildcard characters is available. The wildcard (*) may be used to select any list of items for probing quantities such as voltage or current.

Subcircuit instances can be specified for keywords **V**, **S**, **W**, **VX**, **IX** and **ISUB**. A specific subcircuit node can be referenced or wildcards (*) can be used. For example, the below specifies that the node `X1.1` will be probed.

```
.PROBE TRAN V(X1.1)
```

In the following example all nodes in subcircuit `X1` will be probed.

```
.PROBE TRAN V(X1.*)
```

See also "Using Wildcards in Subcircuit Instances" on page 843. Wildcards, masks, and so on, can be specified for both voltages and currents.

Parameters

- `ANALYSIS`

  Optional. Specifies the analysis type, which can be useful in the case of multiple types of analysis in the *.cir* file. Can be one of the following:

  AC
  Specifies that the probes are required for an AC analysis.

  DC
  Specifies that the probes are required for a DC analysis.

  TRAN
  Specifies that the probes are required for a transient analysis.

  NOISE
  Specifies that the probes are required for a noise analysis. If specified without any other arguments, the noise of all devices will be written to the output file for viewing. For more information on the use of **INOISE** and **ONOISE** for printing the equivalent input noise and output noise, see the example in ".NOISE" on page 744 and "ONOISE" on page 813.

  SSTAC|SSTXF|SSTNOISE|SSTJITTER|TMODSST|FMODSST|TSST|FSST|SSTSTABIL
  Please refer to the Display Command Syntax chapter of the *Eldo RF User's Manual* for more information regarding these RF options.

  Without any analysis specified, the default behavior of Eldo for managing **.PROBE** is to generate the requested probes for all analyses. Since this can generate very large databases in Eldo RF, a default analysis is chosen and a warning printed. For the **.SST** command, the default analysis is FSST. For the **.MODSST** command, the default analysis is FMODSST.

DSP

Displays DSP results. See "DSPxx(label_name)" on page 799 for display options.

- **ALL**

  Specifies that all defwaves, voltages, currents and digital quantities are probed. This is equivalent to specifying **.PROBE W + .PROBE V + .PROBE I + .PROBE S**.

- **I**

  Causes all currents to be saved (node voltages are *not* saved).

- **IX**

  Probes the current flowing in and out of all nodes of all subcircuits. Subcircuit instances can be specified for this parameter.

- **ISUB**

  Equivalent to **.PROBE IX** except that the subcircuit pins are printed instead of indexes, for example **(X1.A)**. Subcircuit instances can be specified for this parameter.

- **PORT**

  Specifies that all voltages and digital quantities are probed. This is equivalent to specifying **.PROBE V + .PROBE SG**.

- **PRINT**

  Probed defwaves, voltages, currents and digital quantities are printed to both the binary output (*.wdb*) and ASCII (*.chi*) files. Equivalent to specifying **.PROBE ALL PRINT**.

- **Q**

  Equivalent to **S**. Causes all digital quantities (VHDL, VHDL-AMS, Verilog and Verilog-AMS) to be saved. Subcircuit instances can be specified for this parameter.

- **S**

  Causes all digital quantities (VHDL, VHDL-AMS, Verilog and Verilog-AMS) to be saved. Subcircuit instances can be specified for this parameter.

- **SG**

  Saves all digital signals.

- **SPARAM**

  Forces Eldo to dump all S parameters in the output file. Only analyses that are in the frequency domain can be specified, for example **AC**, **SSTAC** or **FSST**. For more information on S parameters see "Working with S, Y, Z Parameters" on page 1041.

- **V**

  Causes all node voltages to be saved—this is the default option. Subcircuit instances can be specified for this parameter.

- **VN**

  Only probes node names that are not numbers. The purpose being that nodes named with letters come from the designer, while nodes named with numbers come from an automatic netlister and typically designers wish to only see their own explicitly named nodes. The rule is applied only on the last part of hierarchical node names.

- **VTOP**

  Displays all top level node voltages. The functionality for current is not available (`ITOP` not allowed).

- **VX**

  Probes voltages on all nodes including all nodes of all subcircuits in the netlist. Subcircuit instances can be specified for this parameter.

- **VXN**

  Equivalent to **.PROBE VX** except that the subcircuit pins are printed instead of indexes, for example **(X1.X2.C)**.

- **W**

  Causes all defwaves to be printed in output files. Subcircuit instances can be specified for this parameter.

- **WTOP**

  Probes defwaves at the top level and dumps them to the output file (not defwaves defined in **.SUBCKT** commands).

- **MASK**=mask_name

  Pattern filter to mask specific terminal names (node or device) from global **.PROBE** commands or **.PROBE** commands using wildcards. The rule is applied only to the leaf name of an output specification (voltage or current). Wildcards are allowed. Multiple **MASK** specifications are allowed, but multiple strings are not. Example:

  ```
  .probe tran mask=n* v
  ```

  Masks all terminal nodes or devices that start with n.

  ```
  .probe tran V MASK="I" MASK="J"
  ```

  Masks all nodes with leaf names I or J.

- **EXCEPT**=pattern

  Pattern filter to mask specific output names from global **.PROBE** commands or **.PROBE** commands using wildcards. This rule is applied to the complete output name, hierarchy included, of an output specification (voltage or current). Wildcards are allowed. Multiple **EXCEPT** specifications are allowed, but multiple strings are not. Examples:

  ```
  .probe tran except=x1.x2.n* v
  ```

  Excludes all voltage waveforms whose full pathname starts with x1.x2.n.

```
.probe tran except=N_* v
```

Excludes all voltage waveforms whose full pathname starts with N_.

```
.probe tran except=*x* v
```

Excludes all voltage waveforms related to hierarchical nodes.

```
.probe tran V EXCEPT="X*" EXCEPT="R*"
```

Excludes all nodes with names beginning with X or R.

- **PRINT**

  The probed output will be printed in the *.chi* file, as well as the binary output file (*.wdb*). The first example below shows that voltages for an AC analysis are printed in the *.wdb* and *.chi* files. In the second case, transient probe results are printed in the *.wdb* and *.chi* files.

  ```
  .PROBE AC V PRINT
  .PROBE TRAN PRINT
  ```

Note

To print information to an ASCII file (*.chi*), without printing in the binary output file (*.wdb*), use the **.PRINT** command.

- `alias_name`

  Refers to the wave name in the ASCII and binary output files. The `alias_name` will be the legend displayed inside the wave viewer for the plotted wave as specified by the plot specifications in `OVN`.

- `OVN`

  Output variable name. A list of plot specifications can follow. Mandatory. The syntax for specifying the list of plot specifications to be monitored is provided on "Plot Specifications (OVN)" on page 799. The syntax is separated into a number of sections depending on the device: MOSFET, BJT, JFET, DIODE, with a common section afterwards.

Note

The plot specifications must match the corresponding analysis type, that is, **AC**, **DC**, **TRAN**, or **NOISE**.

`DSPxx(label_name)`

Displays DSP results. Should be specified as part of `OVN`. `xx` can be `DB`, `R`, `I`, `P`, `M`, `GD` (see above).

- **STEP**=`val`

  Performs a sampling of the waveform(s). A point is dumped every `val`. It can only be specified for databases that support multiple x-axes in the same simulation, for example JWDB. Other databases will ignore this parameter.

- **DELTA**=val

  Reduces the number of points dumped into the output files, thus reducing their size. Default value is 0.0. A point with a Y-axis difference less than `val` compared with the previously output point will not be written to the output file. If a point has a Y-axis difference greater than `val` compared with the previously output point then both this point and the previous simulation point will be written to the output file.

Using Wildcards in Subcircuit Instances

The wildcard (*) may be used to select any list of items for probing quantities such as voltage or current. Subcircuit instances can be specified for keywords **V**, **I**, **S**, **W**, **VX**, **IX**, **ISUB** and **POW**. For example, the below specifies that all nodes in subcircuit `X1` will be probed:

```
.PROBE TRAN V(X1.*)
```

The example below will store all currents in devices internal to the Xbias instance:

```
.PROBE TRAN I(Xbias.*)
```

The syntax below can be used to specify wildcards in subcircuit instances:

```
V|I|S|W|VX|IX|ISUB|POW(<instance_name>.*), or
V|I|S|W|VX|IX|ISUB|POW(<instance_name>->*)
```

When specifying parameters **V**, **I**, **VX**, **IX**, **ISUB**, **POW** with a subcircuit instance the **-R** flag can also be specified. Its usage is shown below.

.PROBE TRAN V(X1.***)**

This command will probe all the nodes of subcircuit `X1` but will not probe internal nodes of the subcircuit hierarchy.

.PROBE TRAN -R V(X1.***)**

When specifying the `-R` flag all internal and external nodes of subcircuits beginning with the name `X1` will be probed. (This can also be performed using the **VX** keyword.)

.PROBE TRAN -Rn **V(**X1.***)**

Probe all nodes of subcircuit `X1` and enter inside the `n` level of hierarchy under `X1` for plotting nodes. If `n` is `0` this specifies the top level.

.PROBE TRAN -R LEVEL=n **V(**X1.***)**

Equivalent to **-R**n. Probe all nodes of subcircuit `X1` and enter inside the `n` level of hierarchy under `X1` for plotting nodes. If `n` is `0` this specifies the top level.

Note

.PROBE TRAN V(X1.***)** is equivalent to: **.PROBE TRAN -R**0 **V(**X1.***)**

VX, **IX** and **ISUB** can be used to probe voltages and currents on all subcircuit input nodes within the hierarchy. For example:

.PROBE TRAN IX(X1.***)**

Here the current will be probed across all input nodes of subcircuits beginning with the name X1.

Defwaves and digital quantities can also be probed for subcircuit instances using wildcards.

.PROBE TRAN S(x1.y1->***)**

In this example all internal states (denoted by the syntax "->") of macromodel y1 inside subcircuit x1 will be probed.

.PROBE TRAN W(x1.***)**

This example all defwaves will be probed for subcircuits beginning with the name x1.

If the name used in a **.PROBE** contains square brackets, "[" and "]", these are interpreted as special characters when the wildcard character "*" is used (see Table 10-31). According to Unix convention, these special characters may be overridden by using the delimiter backslash character "\" which removes the special function of the character that immediately follows it. For example, the following only returns names that begin with X1.*, because, according to Unix convention, X[1].* means "all names beginning with "X" and followed by the character "1"":

```
.PROBE tran V(X[1].*)
```

Therefore, to display all nodes beginning with V(X[1].*) either of the following should be used:

```
.option NOCMPUNIX
.PROBE tran V(X[1].*)
```

or:

```
.PROBE tran V(X\[1\].*)
```

The following table contains a listing of the special characters that can be used with the **.PROBE** command.

Table 10-31. Special Characters

Character	Description
*	Matches any sequence of characters.
?	Matches any single character.
[abcd]	Matches any character in the specified set.
[-abcd]	Matches any character in the specified range, for example [A-Z] matches all alphabetical characters.

Table 10-31. Special Characters

Character	Description
[!abcd]	Matches any character not in the specified set, for example `[!0-9]` matches all characters which are not digits.

Signal Monitoring Specifications used with .PROBE Only

V(Xnn.*)** — Monitors the voltage on all of the nodes of subcircuit **X**nn.

S(Xnn.*)** — Monitors the states on all of the nodes of subcircuit **X**nn.

S(Xnn.**Y**nn->*)** — Monitors the states on all of the nodes of macromodel **X**nn.**Y**nn.

Examples

```
*.OPTION description
.option limprobe=1550
...
r1 n30 n400 1k
c1 n490 n610 5p
.probe
```

Specifies that all node voltages will be saved in the <*circuit_name*>*.wdb* file. The circuit has more than 1000 nodes, hence the inclusion of the **LIMPROBE** parameter in the **.OPTION** command.

```
* Circuit description
...
r1 n3 n40 1k
.probe I
```

Specifies that all currents will be saved in the <*circuit_name*>*.wdb* file.

```
.TRAN 1n 10n
.AC dec 10 1k 10k
.probe
```

Specifies that all node voltages for each analysis specified (that is, transient and AC) will be saved in the <*circuit_name*>*.wdb* file. The above **.probe** is equivalent to the following two statements:

```
.probe TRAN V
.probe AC V
```

The following example specifies to probe all nodes of subcircuit `x1`:

```
.probe TRAN V(x1.*)
```

The following example specifies to probe all internal states of macromodel `x1.y1`:

```
.probe TRAN S(x1.y1->*)
```

The following syntax will display all nodes of subcircuit instance `x1` with “2” at the end of the node name:

```
.probe TRAN V(x1.*2)
```

Probe requests such as `I(Q*.C)` as well as `ic(q*)` are accepted. The same applies for extensions B, E and S for BJT, and S, D, G, B for MOSFET.

The following syntax will display nodes in `x1` at hierarchical levels 1 and 2:

```
.probe TRAN v(x1.[!x]*)
.probe TRAN v(x1.x*.[!x]*)
```

In the following example, the wave `v(X1.X2.X3.X4.OUT)` will be plotted inside EZwave, but the legend will be `w1` and not `v(X1.X2.X3.X4.OUT)`:

```
.probe TRAN w1=v(X1.X2.X3.X4.OUT)
```

The following specifies values according to the following: `VX(M1.7)` refers to the intrinsic source voltage, `VX(Q1.3)` refers to the extrinsic emitter voltage, `VX(R1.2)` refers to the ‘second’ pin of resistor `R1`.

```
.probe VX(M1.7)
.probe VX(Q1.3)
.probe VX(R1.2)
```

In the following example, the current on all pins of all `X` instances will be recorded. Wildcards do not have to be specified.

```
.probe IX
```

In the following example, the mask will be exclusively applied to all nodes of the instances `X1` and `X3`. Instance `X2` will *not* be masked and the nodes of `X2` will be probed.

```
.probe tran mask=in V(X1.*) V(X3.*)
.probe tran V(X2.*)
```

The following example shows option **`vxprobe`** used with a **`.PROBE`**. Nodes `v(1)`, `v(2)`, `v(x1.b)`, `v(0)`, `v(x1.a)` and `v(x1.c)` will be probed. Without this option only `v(1)`, `v(2)` and `v(x1.b)` would be probed.

```
.subckt foo a c
r1 a b 1k
r2 b c 1k
.ends foo

v1 1 0 dc 1
x1 1 2 foo
r1 2 0 1

.tran 1n 10n
.probe tran v
.option vxprobe
```

In the following examples, the current on node `2` of voltage source `V1` will be probed, and the current on the first node of resistor `R1` will be probed.

```
.PROBE TRAN I(V1.2)
.PROBE TRAN I(R1.1)
```

The example below stores currents in devices on top:

```
.PROBE TRAN I(*)
```

The example below will store all currents in devices internal to the X1 instance:

```
.PROBE TRAN I(X1.*)
```

The example below shows how to reduce the number of points dumped; points separated by less than 0.1V will not be written in the output files:

```
.PROBE TRAN V(S) delta=0.1
```

.PROBEBUS

Printing of Bus Signals

```
.PROBEBUS BNAME [VTH[1]=VAL [VTH2=VAL]]
.PROBEBUS BNAME[MSB:LSB]|BNAME<MSB:LSB>|BNAMEMSB:LSB
```

This command plots all the bits of a bus, previously defined via the **.SETBUS** and using **.SIGBUS** to send a value. **.SETBUS** can be implicitly declared, providing that BNAME[MSB:LSB] or BNAME<MSB:LSB> or BNAMEMSB:LSB syntax is used.

Parameters

- BNAME

  Bus name, previously defined via the **.SETBUS** command, unless BNAME[MSB:LSB] or BNAME<MSB:LSB> or BNAMEMSB:LSB syntax is used, in which case **.SETBUS** is implicitly declared.

- VTH[1]=VAL

  If a voltage threshold is specified, the bus of an analog signal is plotted as a bus (hexadecimal format), else all the different signals of the bus are plotted separately in the waveform viewer as analog waves. (VTH and VTH1 are equivalent to ensure backwards compatibility.)

- VTH2=VAL

  This can be used to plot the indeterminate value as shown below:

  When only **VTH1** is given:
  If value < **VTH** then logic state 0.
  If value > **VTH** then logic state 1.

  When both **VTH1** and **VTH2** are given:
  If value < **VTH1** then logic state 0.
  If **VTH1** < value < **VTH2** then state X.
  If value > **VTH2** then logic state 1.

- MSB:LSB

  Series of bit names, the most significant bit being defined first. This mechanism can be used to implicitly declare **.SETBUS**.

Examples

In order to display the output of a bus, you can use the following syntax:

```
.PROBEBUS Y<instance_name> -> <vector_name>
```

The following example shows how a second voltage threshold can be used:

```
.probebus foo vth=1 vth2=4
.checkbus OUT_BUS_2 vth=1 vth2=4 base=bin
+ 1000ps 01
+ 7000ps x
+ 12000ps 11
```

```
+ 15000ps 01
+ 19500ps 01
+ 20000ps 10
+ 22000ps 10
+ 30000ps 01
```

Refer to ".CHECKBUS" on page 561 for more information.

.PROBEBUS can be used to implicitly declare **.SETBUS**, providing that `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` or `BNAMEMSB:LSB` syntax is used as shown below:

```
.PROBEBUS SELECT[2:0]
.PROBEBUS DOUT<3:0> VTH=1.65
```

is equivalent to:

```
.SETBUS SELECT SELECT[2] SELECT[1] SELECT[0]
.PROBEBUS SELECT
.SETBUS DOUT DOUT<3> DOUT<2> DOUT<1> DOUT<0>
.PROBEBUS DOUT VTH=1.65
```

The following example shows the most significant bit as 1 and the least significant as 5, where the bus name is `foo`.

```
.PROBEBUS foo1:5 vth=3
```

.PROTECT

Netlist Protection

```
.PROTECT
```

This command is used in conjunction with ".UNPROTECT" on page 925 to exclude a section of a netlist from being copied into the output file. Can be specified to control the beginning of the encryption process with the Eldo Encryption tool.

Example

```
.PROTECT
vin 2 0 ac 0.5
r1 2 3 5k
c3 3 0 0.1p
.ac dec 10 10e+4 10e+8
.plot ac vdb(3)
.UNPROTECT
```

The lines in a netlist between the two commands **`.PROTECT`** and **`.UNPROTECT`** are not printed to the output file.

.PZ

Pole-Zero Analysis

```
.PZ OV
```

This command is used in conjunction with an AC analysis. When included in the input file, a special file called *<circuit_name>.pz* is generated by Eldo. Upon conclusion of the simulation, the Pole Zero post-processor may be activated to extract Pole-Zero data and perform simplification of the results, visualization, and so on.

The poles and zeros are extracted for the transfer function characteristic connecting the input node, specified via the AC analysis command. The output node is specified via the **`.PZ`** command.

Note

This command works only if there is a single AC input source, otherwise an error will be issued.

Parameters

- `OV`

  Request output of the specific node or current through a voltage source to the *<circuit_name>.pz* file. The syntax for voltage or current output is as follows:

- **`I(V`**`xx`**`)`**

  Specifies the current through the voltage source **`V`**`xx`.

- **`V(`**`N1[, N2]`**`)`**

  Specifies the voltage difference between nodes `N1` and `N2`. If `N2` and the preceding comma is omitted, ground is assumed. In the case of subcircuit nodes, `N1` has the form **`(X`**`nn.N1`**`)`** where **`X`**`nn` is the subcircuit instance.

For more details on the Pole-Zero Post-processor, refer to the "Pole-Zero Post-Processor" on page 1121.

.RAMP

Automatic Ramping

```
.RAMP DC VAL [SIMPLIFY]
.RAMP TRAN T1 T2 [SIMPLIFY]
```

This command is used when Eldo encounters difficulties finding a DC operating point with the conventional **.DC** command. There are two options available:

- DC ramping may be performed if the power supplies in a circuit are increased linearly from time 0, by a fixed voltage step. At each step, a DC operating point is searched up to the maximum power supply voltage.
- Transient ramping may be performed if the power supplies are increased linearly from time 0 to `T1`, at which point simulation is continued in the form of a transient analysis. When time `T2` has been reached, a DC operating point is searched for by Eldo.

When carrying out Gmin ramping, or DC ramping, there are usually steep discontinuities. Eldo has various algorithms which may pass over these discontinuities, but they are CPU time consuming. When specifying **SIMPLIFY**, "simplified" Gmin ramping or "simplified" DC ramping will be carried out. In this case, Eldo will attempt to pass over only two discontinuities, before switching to another algorithm, whereupon it should find a DC algorithm which works better on the design. If DC convergence is not achieved by any other algorithms, Eldo will restart Gmin ramping and DC ramping, but will try to pass over all discontinuities.

Transient ramping is a more powerful way of finding a DC operating point than by using DC ramping, but has the disadvantage of being more time consuming. It is recommended that DC ramping methods be tried first and only when everything else fails use transient ramping. The **.RAMP TRAN** emulates a sweep on all of the input signals. On completion of the sweep, Eldo performs a DC analysis to ensure that the solution found at the end of the transient analysis is Steady-State.

If the keyword **SIMPLIFY** is specified at the end of the **.RAMP** command, the last DC analysis will not be performed. If a **.TRAN** command is specified, the transient simulation will start from the values obtained at the end of the **.RAMP TRAN**. This is for cases where the user is not interested in the DC, but wishes to use a time constant for increasing transient input voltages without having to change the input signal of the design.

The DC ramping option, **.RAMP DC** by default, works in conjunction with **.OPTION GRAMP**. If **GRAMP** is set to 0, only **.RAMP DC** is active. When **GRAMP** and **.RAMP DC** are used together, efficiency is improved. See **.OPTION** "GRAMP=VAL" on page 985.

The conductance value that is placed in parallel with all PN junctions of all devices, and drain and source nodes of MOSFET models also changes in accordance with its defined value with the variation of the DC sweep. If DC source ramping *only* is desired, you must set **GRAMP=0** explicitly.

Parameters

- **DC**

  Keyword indicating that DC ramping should be performed.

- **TRAN**

  Keyword indicating that transient ramping should be performed.

- VAL

  Voltage step at which DC ramping is carried out in volts. The ramping process increases the DC source from 0 up to the nominal value. VAL is the largest step that can be used. The default value is 0.1 V.

- **SIMPLIFY**

  Eldo will attempt to pass over two discontinuities, before switching to another algorithm, more suited to the design.

- T1

  The time at which simulation should be continued. The default value is 1 µs.

- T2

  The time at which the DC operating point is searched for. The default value is 10 s.

Related options

"GRAMP=VAL" on page 985, "GNODE" on page 1016, "PSTRAN" on page 1017, "DPTRAN" on page 1017,

.RESTART

Restart Simulation

```
.RESTART FNAME [FILE=WNAME]
.RESTART ["fileBasename"] [NEWEST|LONGEST|TIME=VALUE] [FILE=WNAME]
```

This command restarts a simulation run with information previously saved using either the **.SAVE** or **.TSAVE** command. When a simulation is rerun with **.RESTART**, Eldo looks for a *.iic* or *.sav* file by default. A *.iic* file is the default extension when saving a simulation run with the **.SAVE** or **.TSAVE** command. A *.sav* file is created when a simulation run is interrupted by the user with Ctrl-c. See ".SAVE" on page 857 and ".TSAVE" on page 916.

If there is a mismatch between the actual design and the content of the file when reading back a **.RESTART** file for starting a new TRAN simulation, an error will be returned rather than a warning, and the simulation will stop. That means that the restart command is recommended for a simple transient simulation only.

For more details, refer to the section ".SAVE, .USE & .RESTART" on page 1080 in the Speed and Accuracy chapter.

In the case of **.TRAN** and **.RESTART** associated with a **.STEP**, **.MC**, **.WCASE** or **.NOISETRAN**, the same restart file will be used for all runs, see "Usage of .RESTART and .TRAN with .NOISETRAN, .MC, .WCASE, .STEP" on page 856 for further information.

Parameters

- `FNAME`

  Filename containing simulation information which should be used to restart a simulation run.

- **FILE**=`WNAME`

  Name of file containing the previous *.cou* or *.wdb* file. When this is provided, Eldo concatenates the old and the new binary data files. Binary files can be both *.wdb* and *.cou*. The file format is selected by the file extension, which must match the binary file format. **FILE=** is required.

- `"fileBaseName"`

  Specifies the root name of the *.iic* file that will be used to restart the simulation. Optional. If omitted Eldo will select the *.iic* that has the same root name as the top-netlist that is, if the top-netlist is called *spice-on-top.cir* Eldo will select the file *spice-on-top_*. If Eldo can not find a file with the root name `fileBaseName` it will be assumed that the filename (`FNAME`) has been given.

- **NEWEST**

  The simulation will be restarted with the most recently generated *.iic* file. Optional.

- **LONGEST**

  Restarts the simulation using the *.iic* that was generated at the greatest time point in the simulation. Optional.

- **TIME**=value

  Restarts the simulation from the specified time. If the file does not exist, Eldo will use the file with the closest time. Optional.

Example

```
.restart test.sav file=test_previous.wdb
```

In this example, Eldo will create *test.wdb* as usual, but will first copy *test_previous.wdb* into *test.wdb* before adding new points in the *test.wdb* file. If the *file=test_previous.wdb* string is omitted in the **.RESTART** command, then the first point in the *test.wdb* file will correspond to the time at which simulation actually restarts.

To permit the concatenation of the previous binary output file with the new binary output file, you have to rename it, for example:

Netlist file: *test.cir*
Output file: *test.wdb* (after the first simulation)
Rename this file: *test_previous.wdb*
Relaunch Eldo with:

```
.RESTART test.sav file=test_previous.wdb
```

A new wdb file is created, *test.wdb*, which is the concatenation of the previous and the current simulation file.

```
.RESTART "checkpoint" time=10ns
```

The simulation will be restarted using the *.iic* file that was saved at the time *10ns* and has the root name *checkpoint*.

```
.RESTART LONGEST
```

The simulation will be restarted using the *.iic* file that was saved at the latest time point in the simulation. Eldo will select the *.iic* that has the same root name as the top-design unit that is, if the top-design is called *spice-on-top.cir* Eldo will select the file *spice-on-top_1.000000E-08.iic*.

Usage of .RESTART and .AC

If you have in your input deck both of the following commands:

- **.RESTART** <file_name>, where file_name comes from the results of a TRAN simulation
- **.AC ... UIC** command

then the **.RESTART** will be interpreted as:

```
.USE <file_name> OVERWRITE_INPUT
```

Refer to the **.AC** command **UIC** parameter, see "UIC" on page 540 for more information.

Usage of .RESTART and .TRAN with .NOISETRAN, .MC, .WCASE, .STEP

In the case of **.TRAN** and **.RESTART** associated with a **.STEP**, **.MC**, **.WCASE** or **.NOISETRAN**, the same restart file will be used for all runs, for example:

```
.MC 5
.TRAN
.SAVE TIME = ...
.END
```

In this example you would assume that six restart files would be generated (five MC and one nominal), and that in subsequent commands:

```
.MC 5
.TRAN
.RESTART
.END
```

each run would use its "restart file ancestor".

However, this is not how it works. Eldo will use only the latest "restart file" which has been generated for all runs, and unless explicitly specified, Eldo will create a restart file only for the first STEP run or the nominal MC run.

.SAVE

Save Simulation Run

```
.SAVE [[FILE=]FNAME] DC|END|TIME=VAL1 [REPEAT [ALT|SEQ]]
+ [TEMP=VAL2] [STEP=VAL3] [TYPE=NODESET|IC] [LEVEL=ALL|TOP] [CARLO=index]
```

Writes information at specific times during simulation to a file `FNAME`. If no filename is given, results are saved in *<circuit_name>.iic*. To restart a simulation run with this information saved, use the **`.RESTART`** command. See ".RESTART" on page 854.

For more details, refer to the section ".SAVE, .USE & .RESTART" on page 1080 in the Speed and Accuracy chapter.

For saving a simulation run at multiple time points, see ".TSAVE" on page 916.

A warning will be printed if the **`.SAVE`** command cannot be performed.

Parameters

- **`DC`**

  Keyword indicating that the DC part of the simulation should be saved in `FNAME`. These results may at a later time be used as **`.NODESET`**, **`.IC`**, or **`.GUESS`** values for another simulation via the **`.USE`** command.

- **`END`**

  Keyword indicating that the simulation results should be saved after transient analysis has been completed in `FNAME`. These results may then be used as re-starting values of another simulation via the **`.RESTART`** command.

- **`TIME`**

  The value of this keyword is the discrete instant in time after the start of simulation that the simulation results should be saved in `FNAME`. These results may then be used as re-starting values of another simulation via the **`.RESTART`** command. This option may also be used in conjunction with the **`REPEAT`** parameter (See below).

- `VAL1`

  Time, in seconds, at which the simulation results should be saved to `FNAME`. A number, expression, or parameter can be specified.

Note

The **`TIME`** option has the same meaning as the **`.OPTION SAVETIME`**`=VAL` command, except that the former specifies simulation time and the latter specifies actual CPU time in hours.

- **FILE**

  Optional; used to set the name of the output file.

- FNAME

  Filename and extension into which simulation information is written. Filename is alphanumeric but must start with an alpha (letter).

- **REPEAT**

  Keyword, used in conjunction with the **TIME**, **ALT** and **SEQ** options. When selected, Eldo saves the status of the simulation at every time interval VAL1 to FNAME in accordance with the **ALT** and **SEQ** parameters described below. **ALT** is the default if neither are specified.

- **ALT**

  Used in conjunction with **REPEAT** to create the files FNAME.a, FNAME.b, FNAME.a, ... in an alternating order, thus overwriting the previous file of the same extension and resulting in only two files FNAME.a and FNAME.b being created. Default when **REPEAT** is specified.

- **SEQ**

  Used in conjunction with **REPEAT** to create the files FNAME.1, FNAME.2, FNAME.3, ... FNAME.N, in a sequential manner, thus creating N files depending on the values of **TIME** and **REPEAT**.

- **TEMP**

  Keyword indicating that data should be saved to FNAME depending on temperature.

- VAL2

  Temperature, in degrees, at which the circuit data should be saved to FNAME.

- **STEP**

  Keyword, used in conjunction with the **.STEP PARAM** command. When selected, Eldo saves the status of the simulation to FNAME if the parameter value specified in the **.STEP** command is equal to a specified value.

- VAL3

  Value at which the status of the simulation should be saved to FNAME when equal to the parameter value specified in the **.STEP** command.

- **TYPE**

  Used to set the type of information being dumped. Eldo will generate a Spice-format file which can be loaded with the commands **.LOAD** or **.INCLUDE** for subsequent runs.

  If NODESET is specified, then when Eldo reads back the file from a **.LOAD** command, the node voltage information is consider as being equivalent to a **.NODESET** command.

  If IC is specified, then the node voltage information is considered as being equivalent to a **.IC** command.

Specifying **TYPE** means the file will contain less information than the native Eldo save file (*.iic*). However, it can be re-used on other Spice-like simulators.

- **LEVEL=ALL**

  All nodes have to be dumped in the file (default).

- **LEVEL=TOP**

  Only top nodes (that is, those which are not inside a SUBCKT) are dumped in the file.

> The previous three parameters work in conjunction with the **.LOAD** command, see ".LOAD" on page 698.

- **CARLO**=index

  Save simulation information for a specific run (index) of a Monte Carlo analysis. This is useful for debugging purposes, as it would be possible to restart a specific run from a series of Monte Carlo simulations. The DC value would be re-injected into a single Monte Carlo run (**IRUN**=index of the .MC command).

Related options

SAVETIME (see "SAVETIME=VAL" on page 1015)

Examples

```
.save test.ddt dc
```

Specifies that DC analysis results of the current circuit simulation should be written to the *test.ddt* file.

```
r1 1 2 p2
...
.step param p2 1k 5k 1k
.save status.ccf step=4k
```

Causes status of current simulation to be saved in *status.ccf* file when parameter p2 reaches 4kΩ.

.SCALE

Automatic Scaling of Active Devices

```
.SC[ALE] ELTYPE KEYWORD VALUE [KEYWORD VALUE ...]
+ [ELEMENTS ALL|EXCEPT] [ELNAME1 ELNAME2 ...] [(ELNAME1 ELNAME2)]]
.SC[ALE] ELTYPE KEYWORD VALUE [KEYWORD VALUE ...]
+ MODELS MODNAME1 [MODNAME2 ...]

.SC[ALE] MODTYPE KEYWORD VALUE [KEYWORD VALUE ....]
+ [MODELS ALL|EXCEPT] [MODNAME1 MODNAME2] [(MODNAME1 MODNAME2...)]]

.SC[ALE] P FACTOR=VALUE [SUBCKT=SUBNAME] [INST=INSTNAME]
+ [PARAMS ALL|EXCEPT] [PARAM1 PARAM2 ...] [(PARAM11 PARAM22)]]
```

This command scales device and model parameters of active devices automatically.

The first two syntax shown above are the general forms for devices. The third syntax is the general form for device models. The last syntax is an additional feature for parameter scaling.

Parameters

- `ELTYPE`

  Type of element. One of the following element types:

`D [LEVEL=n]`	Diode
`Q [LEVEL=n]`	BJT
`J [LEVEL=n]`	JFET
`M [LEVEL=n]`	Level-n MOSFET

- `MODTYPE`

  Type of model. One of the following model types:

`DM [LEVEL=n]`	Diode model
`NPN [LEVEL=n]`	npn BJT model
`PNP [LEVEL=n]`	pnp BJT model
`NJF [LEVEL=n]`	n-channel JFET model
`PJF [LEVEL=n]`	p-channel JFET model
`NMOS [LEVEL=n]`	n-channel level-n MOS model
`PMOS [LEVEL=n]`	p-channel level-n MOS model

- `KEYWORD`

  Any valid element or model parameter or keyword.

- `ELEMENTS`

  Keyword indicating the end of a keyword/value pair list for elements.

- `PARAMS`

  Keyword indicating parameter scaling.

- `ALL`

  Keyword requesting scaling of all elements or models.

- `EXCEPT`

  Keyword requesting scaling of all elements or models with the exception of the ones listed.

Limitation

The element parameters or devices are only scaled at the parsing level.

If a **`.DC`** is used on a device element with a scale factor, the device element takes only the value specified by the DC analysis. There is no scaling effect.

.SELECT_DSPF_ON_NODE

Select DSPF on Specified Node

```
.SELECT_DSPF_ON_NODE {NODE}
```

Used to force Eldo to select parasitic elements on a specified node when a DSPF file has been included using the **.DSPF_INCLUDE** command.

This command is ignored when **DEV=DSPF** is specified on the **.DSPF_INCLUDE** command.

Parameters

- NODE

  Specifies node(s) for which the parasitics given in the DSPF file will be selected.

Example

```
.DSPF_INCLUDE testspf.spf
.SELECT_DSPF_ON_NODE n1 n2
```

For nodes n1 and n2 parasitics as specified in DSPF file *testspf.spf* will be selected.

.SENS

DC Sensitivity Analysis

```
.SENS OVN {OVN}
```

This command causes a DC sensitivity analysis to be performed. By linearizing the circuit around a bias point, the sensitivities of each of the output variables in relation to all the device values and the model parameters are calculated and output.

This Eldo command is restricted to DC sensitivity analysis. For AC sensitivity analysis, see ".SENSAC" on page 865. For TRAN or SST sensitivity analysis, see ".WCASE" on page 934.

The DC sensitivity analysis will be performed only for the first DC of a DCSWEEP.

Note

For MOS, only sensitivity with respect to L and W is implemented. For BJT, Diodes and JFET, sensitivity is implemented with respect to all model parameters.

Device sensitivities are provided for the following types:

- Resistors.
- Independent Voltage and Current Sources.
- Voltage and Current Controlled Switches.
- Diodes and Bipolar Transistors.

Parameters

- `OVN`

  Output variable name. Request output of the specific node or current through a voltage source to the *<circuit_name>.chi* file. Syntax for voltage or current output is as follows:

- **`V(`**`N1[, N2]`**`)`**

  Specifies the voltage difference between nodes `N1` and `N2`. If only `N1` is specified, ground is assumed for the second node.

- **`I(V`**`xx[, `**`V`**`yy]`**`)`**

  Specifies the current difference between voltage sources **`V`**`xx` and **`V`**`yy`. If only **`V`**`xx` is specified, current through **`V`**`xx` is printed.

Example

```
.sens v(2) i(vcc)
```

Specifies a sensitivity analysis to be performed on the voltage at node `2` and on the current through the voltage source `vcc`. The results of the analysis are listed in the *<circuit_name>.chi* file.

An example of this type of analysis can be found in “Tutorials” on page 1341.

.SENSAC

AC Sensitivity Analysis

```
.SENSAC OVN {OVN} FREQ=val1[{,val2}] [SORT_REL] [SORT_ABS] [SORT_MAX]
```

This command causes an AC sensitivity analysis to be performed. By linearizing the circuit around a bias point, the sensitivities of each of the output variables in relation to R, L, C devices present in the design are calculated and written to the output *.chi* file.

This Eldo command is restricted to AC sensitivity analysis. For DC sensitivity analysis, see ".SENS" on page 863. For TRAN or SST sensitivity analysis, see ".WCASE" on page 934.

Device sensitivities are provided for the following types:

- Resistors
- Capacitors
- Inductors

The output is sorted from the most sensitive to the least sensitive.

Note

This command was previously named **.ACSENSRLC**.

Parameters

- OVN

  Output variable name. Request output of the specific node to the *<circuit_name>.chi* file. Syntax for voltage or current output is as follows:

  V(N1[, N2]**)**

  Specifies the voltage difference between nodes N1 and N2. If only N1 is specified, ground is assumed for the second node.

  I(Vxx[, **V**yy]**)**

  Specifies the current difference between voltage sources **V**xx and **V**yy. If only **V**xx is specified, current through **V**xx is printed.

- FREQ

  List of frequency values at which the sensitivities must be computed.

- SORT_REL

  Used to filter the output. Only devices with a contribution to the output greater than the value: SORT_REL×<MAX_variation_on_output> will be listed. The default value of SORT_REL is 0.001, which means that all devices contributing to more than 1/1000 of the maximum variation will be listed.

- `SORT_ABS`

  Used to filter the output. Specifies the absolute threshold, below which contributors will not be listed. Default value is 0.

- `SORT_NBMAX`

  Used to filter the output. Allows the user to limit the list of contributors to a certain value. By default, all contributors are listed.

Note

The sorting parameters are additive, that is, their respective effects are cumulative. For example, to create a report with only devices contributing to 10% or more of the maximum output deviation and have 15 contributors at most you would specify:

```
SORT_REL=0.1
SORT_NBMAX=15
```

Example

```
v1 1 0 ac 1
rv1 1 0 1
e1 2 0 1 0 1000
r1 2 0 1k
r2 2 3 1k

r3 3 4 1k
c1 4 0 100n
r4 4 0 1k

.ac dec 20 10 1giga

.extract ac yval(vm(3),10k)
.sensac v(3) freq=10k
```

Specifies a sensitivity analysis to be performed on the voltage at node `3` at a frequency of 10kHz. The results of the analysis are listed in the *<circuit_name>.chi* file, example results:

```
.SENSAC  V(3) SORT_REL = 1.000000e-03

Analysis results at  1.0000E+04 Hertz:

         ELEMENT      ELEMENT       ELEMENT       NORMALIZED
          NAME         VALUE      SENSITIVITY    SENSITIVITY
                                  (VOLTS/UNIT) (VOLTS/PERCENT)

    C on  4          1.000E-07     -1.94E+08     -1.94E-01
```

.SENSPARAM

Sensitivity Analysis

```
.SENSPARAM sub[ckt]=subckt_name param=parameter_list
+ [var[iation]=value] [inst[ance]=instance_list]
+ [sort=inc[reasing] | dec[reasing] | alpha[betical]]
+ [sort_nbmax=value] [sort_abs=value | sort_rel=value]
```

This command activates a sensitivity analysis of extracts versus subcircuit parameters. Designers need sensitivity information for design parameters (parameters they can act upon). This command computes the sensitivity of voltages/currents and extracts to these design parameters.

The first simulation is the nominal simulation, using nominal values for the parameters. The following simulations are sensitivity analysis runs where Eldo applies a small variation to one of the P parameters, keeping all other to their nominal value, and computes the sensitivity of the targets relative to the current parameter. Therefore, there are 1 + P simulations.

The sensitivity results represent the variation of the extracted value for a variation of 1% of the parameter value.

Parameters

- `sub[ckt]`

  A subcircuit name.

- `param`

  Defines the list of parameters of subcircuit `subckt_name` for which the sensitivity must be calculated.

- `var[iation]`

  Defines the variation value for the sensitivity parameters. Default is 0.1%.

- `ins[tance]`

  Defines a list of instances of subcircuit `subckt_name`. The sensitivity analysis will be restricted to parameters of these instances. This keyword supports standard wildcards: * and ?. For example:

  ```
  .sensparam subckt=resi instance=x2.* param=a,b variation=7%
  ```

  means that all parameters a and b of instances of subcircuit resi inside instance X2 will be subject to sensitivity.

  ```
  .sensparam subckt=resi instance=x2 param=c,e variation=0.03
  ```

  means that parameters c and e of instance x2 of subcircuit resi only will be subject to sensitivity.

- `sort`

  Defines the ordering method which will be used to print sensitivity results. Default is alphabetical.

- `sort_nbmax`

  Only the specified n first contributions will be printed.

- `sort_abs`

  Only the contributions greater than the given value in absolute will be printed.

- `sort_rel`

  Only the contributions greater than the given value in relative will be printed.

Note

If several **`.SENSPARAM`** commands are specified in the netlist, only the last value for keywords **`sort`**, **`sort_nbmax`**, **`sort_abs`** and **`sort_rel`** will be retained.

Example

```
* .sensparam example
.subckt resi a b param: a=1 b=1 c=1 d=1 e=1
R1 a b r={a+2*b+3*c+4*d+5*e}
.ends resi

.subckt div a b param: a=1 b=1 c=1 d=1 e=1
x1 a 1 resi a=a b=b c=c d=d e=e
x2 1 b resi a=a b=b c=c d=d e=e
.ends div

v1 1 0 sin(0 1 0.5g)
x1 1 2 resi a=2 b=1 c=5 d=6 e=10
X2 1 2 resi a=10 b=2 c=6 d=3 e=1
X3 1 2 div
r2 2 0 1

.sensparam subckt=resi instance=x3.* param=a,b variation=2%
.sensparam subckt=resi instance=x2 param=c,e variation=0.03
.sensparam subckt=resi param=d sort_abs=4e-4

.tran 1n 15n

.extract tran yval(i(r2),5n)
.end
```

The three **`.SENSPARAM`** commands in the above example define which parameters will be used for sensitivity analysis. The first command defines that parameters a and b of all instances of subcircuit resi* inside instance X3 will have a variation of 2% applied to their nominal values. The second command defines that parameters c and e of top instance X2 will have a variation of 3% applied to their nominal values. The third command defines that parameter d of all instances of subcircuit resi will have a default variation of 0.1% applied, that contributions lower than 40e-9 will not be printed, and are sorted in decreasing order.

The sensitivity results represent the variation of the extracted value for a variation of 1% of the parameter value. See example results below.

```
YVAL(I(R2),5N) NOM:  5.2297E-05
*| Normalized         Extract          Parameter      Parameter    Parameter
*| Sensitivity (%/%) Variation (/%) Nominal Value  Variation    Name
*|  1.1039E-01       -5.7732E-08    6.00000e+00    3.000e+00%   X2.C
*|  7.2816E-02       -3.8081E-08    3.00000e+00    1.000e-01%   X2.D
*|  6.4746E-02       -3.3860E-08    1.00000e+00    1.000e-01%   X3.X1.D
*|  6.4746E-02       -3.3860E-08    1.00000e+00    1.000e-01%   X3.X2.D
*|  4.0429E-02       -2.1143E-08    6.00000e+00    1.000e-01%   X1.D
```

.SETBUS

Create Bus

```
.SETBUS BNAME PN {PN}
```

This command creates a bus BNAME with a number of bits PN.

Note

Bus signals are defined using the **.SIGBUS** command.

Parameters

- BNAME

  Bus name.

- PN

  Bit name, the most significant bit being defined first. Groups of bits can be specified as a bit range, and Eldo will expand the bus. Suppose i and j are two integers, the expansion rule is as follows:

  If i < j, then n[i:j] is expanded into n[i], n[i+1], n[i+2] ... n[j]

  If i > j, then n[i:j] is expanded into n[i], n[i-1], n[i-2] ... n[j]

  To disable the automatic bus expansion, specify option **NOSETBUSEXPAND**.

Related option

NOSETBUSEXPAND (see "NOSETBUSEXPAND" on page 956)

Example

```
.setbus bus1 p5 p4 p3 p2 p1 p0
```

Specifies a bus bus1 with the pins p5, p4, p3, p2, p1 and p0. Pin p5 is the most significant bit.

The following example shows bus expansion:

```
.setbus foo a[4:2]
```

is equivalent to:

```
.setbus foo a[4] a[3] a[2]
```

The following is another example of bus expansion with multiple bus specifications:

```
.setbus bus2 a[4:2] b1 b[0] a[9:6]
```

Specifies a bus bus2 with a[4], a[3], a[2], b1, b[0], a[9], a[8], a[7] and a[6].

.SETKEY

Set Reliability Model Key (Password)

```
.SETKEY [MODEL=model_name] KEY=key_value
```

This reliability analysis command defines a key (password), which is associated to a specific model or to all the models of a library.

For the complete description of this command and information on all reliability commands, see the separate chapter Reliability Simulation.

.SETSOA

Set Safe Operating Area

```
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ E {EXPRESSION=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ E IF(EXPR) THEN({PARAM=(MIN,MAX[,XAXIS])})
+ ELSE({PARAM=(MIN,MAX[,XAXIS])}) ENDIF
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ E SUBCKT=subckt_list {PARAM=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ D DNAME [SUBCKT=subckt_list|INST=inst_list] {PARAM=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ D DNAME [SUBCKT=subckt_list|INST=inst_list]
+ IF(EXPR) THEN({PARAM=(MIN, MAX[, XAXIS])})
+ ELSE({PARAM=(MIN, MAX[, XAXIS])}) ENDIF
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ M MNAME [SUBCKT=subckt_list|INST=inst_list] {PARAM=(MIN,MAX[,XAXIS])}
.SETSOA [LABEL="STRING"] [ANALYSIS] [SOACODE=val]
+ M MNAME [SUBCKT=subckt_list|INST=inst_list]
+ IF(EXPR) THEN({PARAM=(MIN, MAX[, XAXIS])})
+ ELSE({PARAM=(MIN, MAX[, XAXIS])}) ENDIF
```

Specifies the safe operating area limits for device parameters (D), model parameters (M) and Eldo expressions (E).

Caution

Eldo issues a warning at run time whenever a safe operating limit is violated.

Devices may be selected using either the device name or its model name (if one exists). If limits are specified using both device and model names for a component, then both results are produced. There is no priority selection, and so the particular device must respect all conditions specified to generate no warning (regardless of whether these are the same or different).

Limits set using **.SETSOA** are checked when the **.CHECKSOA** command is used, see ".CHECKSOA" on page 564.

This command applies to all types of analysis, as specified, and may also be used in conjunction with **.TEMP**, **.STEP**, **.MC**, and **.WCASE** commands. For the last two cases, only a check of the typical case is done. However, if the **.MC** analysis statement contains the **ALL** parameter, the **.CHECKSOA** will be activated for each Monte Carlo run.

The IF statement used in conjunction with the keywords THEN, ELSE, ENDIF may be used with expressions to specify conditions for when a SOA should be checked.

It is possible inside SOA expressions to refer to device instance or model parameters via its parameter name, such as: `D(*,<parameter_name>)` or `M(*,<parameter_name>)`. The wildcard character `*` specifies that Eldo will search for the parameter in the device or model specified in `DNAME` or `MNAME` respectively. Model parameters can be specified for devices and device parameters can be specified for models.

At the top level, a list of subcircuits or instances at a lower level of hierarchy can be specified with devices or models.

The name of a model can be prefixed with X instance or subcircuit names. Therefore the SOA model name should contain the wildcard character `*` if the SOA should also apply to all of the extended models. For example, if a model attached to XSUBCKT.M1 is NCH.10, and a model attached to XT.MAIN is XT.NCH.10, the statement .SETSOA M NCH will detect that XSUBCKT.M1 has a model valid for checking, not XT.MAIN. For improved name matching, the statement **`.SETSOA M`** `'*NCH*'` should be specified.

This command also accepts an optional label as the first argument. The label will appear in the ASCII output file, which can be useful for readability of this file.

You can use the E(xpressions) parameter to specify a named subcircuit at any part of the hierarchy and set safe operating limits.

A code can be assigned to any SOA check, using the **`SOACODE`** parameter. Based on this code you can filter SOA checks to enable/disable in the **`.CHECKSOA`** command.

Safe Operating Area warnings can be searched automatically using the following AWK script:

```
#!/bin/sh
awk '
  BEGIN {
     found = 0
        }
        ($2 ~ /SOA/ && $3 ~ /INFORMATION/) {
          found=1
        }
        (found==1){
          if ($1 == "*|")
            print $0
        }
      ' $*
```

Parameters

- **`LABEL`**`="<STRING>"`

  Specified as the first argument. The label will appear in the ASCII output file, which can be useful for readability of this file.

- `ANALYSIS`

  Optional. By default, it depends on the analysis specified in the netlist. Can be one of the following:

  `AC`

  Specifies that the checks are required for an AC analysis.

DC

Specifies that the checks are required for a DC analysis.

TRAN

Specifies that the checks are required for a transient analysis.

DCSWEEP

Specifies that the checks are required for a DCSWEEP analysis.

FOUR

Specifies that the checks are required for an FFT analysis.

TMODSST|**TSST**|**FSST**

Specifies that the checks are required for an RF analysis.

- **SOACODE**=val

  A code assigned to an SOA check. Based on this code you can filter SOA checks to enable/disable in the **.CHECKSOA** command. The code should be a positive integer quantity.

- EXPRESSION

  An expression whose calculated value is to be checked. This can use the same syntax as described in the **.EXTRACT** command, section "FUNCTION" on page 645, for example:

  ```
  .SETSOA E xup(v(out),4,0,100n,1)
  + -xup(v(out),1,0,100n,1) = (*,3n)
  ```

- DNAME

  Name of a device whose parameter(s) are to be checked. The wildcard character * can be used in a device name to specify that all devices of the same type should be checked (for example .SETSOA D r*... will check all resistors in the netlist).

- MNAME

  Name of a model whose parameter(s) are to be checked. The following device categories can also be specified: NMOS, PMOS, NPN, PNP, NJF, PJF, D, for example:

  ```
  .SETSOA M NPN ...
  ```

  indicates all NPN devices will be checked.

- PARAM

  Name of the parameter to be checked for example:
  IB, **IC**, **IE**, **IS**, **VBE**, **VBC**, **VBS**, **VCE**, **VCS**, **VES**, **POW**, **VC**, **VS**, **VB**, **VE** for a bipolar transistor.
  IG, **IS**, **ID**, **IB**, **VGD**, **VGS**, **VGB**, **VBS**, **VBD**, **VDS**, **POW**, **VS**, **VD**, **VG**, **VB** for a MOS/JFET.
  VPOS, **VNEG** (for voltage on positive/negative pin), **VDIP**, **I**, for a dipole.
  POW for power.
  To access the **VTH** value, use **VT(**device_name**)**,
  to access the **VDSAT** value, use **VDSS(**device_name**)**.
  The wildcard character * can be used for the instance specifier in conjunction with device

categories, for example: `VGS(*), VCE(*)`
Expressions are also allowed, for example:
`.SETSOA M NMOS VGS(*) + VDS(*) = (0,*)`

- `MIN`

  Minimum value of expression/parameter to check. SOA limits can also be defined using **`.DATA`** statements for expressions.

- `MAX`

  Maximum value of expression/parameter to check. SOA limits can also be defined using **`.DATA`** statements for expressions.

Note
Expressions are also accepted in the `MIN/MAX` boundaries, together with parameters. Specifying `*` for a `MIN/MAX` boundary means that the appropriate limit should not be checked.

- `XAXIS`

  Specifies the x-axis length that must be achieved before SOA warnings are printed.

- `IF(EXPR)`

  Defines an IF statement where `EXPR` is the expression for the IF condition. The following operators are allowed:

  `||` for `OR`
  `&&` for `AND`
  `==` for `EQUAL`
  `<` or `<=` for `INFERIOR` and `INFERIOR or EQUAL`
  `>` or `>=` for `SUPERIOR` and `SUPERIOR or EQUAL`

  IF statements can be nested. Only the following parameters are allowed in IF expressions: V( ), I( ), P( ), E( ), M( ), and EM( ).

- `THEN`

  Defines a THEN statement. Used in conjunction with `IF`. The user can define a parameter(s) to be checked when the IF statement is true.

- `ELSE`

  Defines an ELSE statement, Used in conjunction with `IF` and `ELSE`. The user can define a parameter(s) to be checked if the IF statement is false.

- **`SUBCKT`**`=subckt_list`

  Specifies a list of subcircuits to be checked. Valid for the D and M types.

- **`INST`**`=inst_list`

  Specifies a list of instances to be checked. Valid for the D and M types.

Examples

```
SOA check
.width out=80
.model n npn
r1 in out 10k
c1 out 0 1p
q1 c b 0 0 n
rl c vdd 1k
vin in 0 pwl 0 0 1n 10 20n 10 21n 100
vdd vdd 0 5
vb b 0 pwl 0 –1 40n 1
.setsoa d r1 i=(*, 3m)
.setsoa m n ic=(–1u, 3m)
.setsoa e IC(q1)/IB(q1)=(*, 100)
.checksoa
.option eps=1u
.tran 1n 40n
.plot tran v(out)
.plot tran i(r1) ib(q1) ic(q1)
.end
```

Caution

This produces the following warning on the screen (or in the *.log* file, if you are simulating in background):
***WARNING: SOA DETECTION: See output file for details

The following results will be obtained in the *.chi* file:

```
1*******5–Feb–2001 **********  Eldo v5.4    **********10:47:21******
0SOA CHECK
0****     SOA INFORMATION           TEMPERATURE =   27.000 DEG C
0*************************************************************
*| R1:
*|  I(R1)
*|    X AXIS WINDOW: [ 20.32351N   31.73637N ] Value superior to 3.000000e-
03
*|  IC(Q1)/IB(Q1)
*|    X AXIS WINDOW: [ 28.42081N   32.91932N ] Value superior to
1.000000e+02
*| Q1:
*|  IC(Q1)
*|    X AXIS WINDOW: [ 35.33011N   40.00000N ] Value superior to 3.000000e-
03
```

The following is an example of using the optional 3rd boundary `XAXIS` parameter:

```
.SETSOA D M1 VGS = (1, 2, 3u) VGS = (0, 4, 1u)
```

this means that when `VGS` is outside the limits `[1, 2]` for a time-period greater than or equal to 3µs, OR when `VGS` is outside the limits `[0, 4]` for a time-period greater than or equal to 3µs a warning is given.

The optional label specification can be used, as shown in the example below:

```
.setsoa label="My severe error" m ndig
+ Vg(*)-Vb(*)=(-15.0,15.0)
.setsoa label="My warning" m ndig
+ Vg(*)-Vb(*)=(-22.3,22.3)
```

The ASCII output file, with the label appearing, would look similar to the following:

```
*| M#$I10:
*|  VG(*)-VB(*) LABEL="My severe error"
*|    X AXIS WINDOW: [ 110.00000N  1.00000U  ]
*|    Value inferior to -1.500000e+01
*|  VG(*)-VB(*) LABEL="My warning"
*|    X AXIS WINDOW: [ 117.30000N  1.00000U  ]
*|    Value inferior to -2.230000e+01
```

The following example shows how the boundary specifications can be used inside SOA.

```
.SETSOA E xup(v(out),4,0,100n,1)
+ -xup(v(out),1,0,100n,1) = (*,3n)
```

Here, SOA notification will appear if the extracted value is higher than 3n.

The following example shows how subcircuit parameters can be used inside SOA.

```
.SUBCKT TEST  A B
R1 A B 1k
.SETSOA D R1 V=(-CH,CH)
.ENDS TEST

X1 1 0 TEST CH=25
Vd 1 0 30
.DC vd 10 30 0.1
.PLOT DC i(Vd)
.CHECKSOA

.END
```

The example below shows multiple expressions can be used in a **.SETSOA** command.

```
.SETSOA E IC(q1)/IB(q1)=(*, 100)
+ xup(v(out),4,0,100n,1) = (*,4n)
```

The following examples show IF statements used in **.SETSOA** commands.

```
.SETSOA M NMOS IF(D(*,W) < 10U) THEN VGS(*) = (*,1.2) ENDIF
```

In the example above, the `VGS` parameter on all NMOS transistors with a device parameter `W` of less than 10µm will be checked with respect to a maximum value of 1.2.

```
.SETSOA M NMOS IF((Id(*) >= 0.1u)&&(Vgs(*)>(VT(*)-0.3)))
+ THEN Vds(*) - VDSS(*) = (0,*) ENDIF
```

In the example above, the IF statement sets up the condition that for all NMOS transistors that have a value of parameter `Id` greater than or equal to 0.1 *and* parameter `Vgs` is greater than the value computed by `VT-0.3`, then the value calculated by `Vds(*)-VDSS(*)` will be checked with respect to a minimum value of 0.

```
.SETSOA E IF(P1>0) THEN IF(P2>0) THEN V(1)=(*,0.5) ENDIF ENDIF
```

In the example above, if parameter `P1` is greater than zero, then the second IF statement will be performed. The second IF statement defines that parameter `V(1)` will be checked with respect to a maximum value of 0.5 if parameter `P2` is greater than zero.

```
.DATA vout
+ tt    voutmax    voutmin
+ 0     0.1        0.0
+ 80n   0.1        0.0
+ 120n  5.0        4.5
+ 200n  5.0        4.5
.ENDDATA
.SETSOA label=vout_correct E v(OUT)=[vout(voutmin),vout(voutmax)]
```

The example above shows how SOA limits can be defined using **`.DATA`** commands.

The following example shows how SOA limits can be checked from the top level using wildcards and across instance/subcircuit lists at a lower level of hierarchy.

```
.SUBCKT NHVT D G S B PARAM: W=1.0 L=0.5 xl=0
.MODEL NHVT NMOS LEVEL=53 XL = p1
M1 D G S B NHVT W=W L=L
.ENDS
X1 D G S B NHVT xl = 0.1u
X2 D G S B NHVT
X3 D G S B NHVT
```

Eldo will create internally two models: one named *X1.NHVT*, used by device *X1.M1*, and another one named *NHVT.NHVT*, shared by *X2.M1* and *X3.M1*. If you wish to perform some SOA checks on the model *NHVT*, specify:

```
.SETSOA M '*.NHVT' VDS=(-1.2,1.2)
```

An equivalent syntax is:

```
.SETSOA M NHVT SUBCKT=NHVT VDS=(-1.2,1.2)
```

It will check for *VDS* on the three instances *X1.M1*, *X2.M1*, and *X3.M1*.

Specifying the following, Eldo will check only for instance *X1.M1*:

```
.SETSOA M NHVT INST=X1 VDS=(-1.2,1.2)
```

The following shows how to check subcircuits in your netlist using the E(xpression) syntax:

```
.subckt mysub n1 n2
Rend1 n1 3 1k
Rp 3 4 2k
Rend2 4 5 3k
C2 5 n2 .1uf
.ends

.setsoa E subckt=mysub Isub(n1) = (1m, 2m)
.setsoa E subckt=mysub v(n1) = (1, 2)
```

```
.setsoa E subckt=mysub v(n1,n2) = (1, 2)
```

.SIGBUS

Set Bus Signal

```
.SIGBUS BNAME|BNAMEMSB:LSB|BNAME[MSB:LSB]|BNAME<MSB:LSB>
+ [VHI=VAL] [VLO=VAL] [TFALL=VAL] [TRISE=VAL]
+ [BASE=OCT|DEC|BIN|HEX] [SIGNED=NONE|1COMP|2COMP]
+ TN VAL {TN VAL} [P]
.SIGBUS BNAME|BNAMEMSB:LSB|BNAME[MSB:LSB]|BNAME<MSB:LSB>
+ [VHI=VAL] [VLO=VAL] [TFALL=VAL] [TRISE=VAL] [THOLD=VAL] [TDELAY=VAL]
+ [BASE=OCT|DEC|BIN|HEX] [SIGNED=NONE|1COMP|2COMP]
+ PATTERN $(PAT) {$(PAT)} |VAL {VAL} [Z]
.SIGBUS BNAME|BNAMEMSB:LSB|BNAME[MSB:LSB]|BNAME<MSB:LSB>
+ [VHI=VAL] [VLO=VAL] [TFALL=VAL] [TRISE=VAL] [THOLD=VAL] [TDELAY=VAL]
+ [BASE=OCT|DEC|BIN|HEX] [SIGNED=NONE|1COMP|2COMP]
+ FILE=FILE
```

This command sets signals on a bus `BNAME` that has been previously defined via the **`.SETBUS`** command. **`.SETBUS`** can be implicitly declared, providing that `BNAMEMSB:LSB`, `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` syntax is used.

In its first form shown above, **`.SIGBUS`** defines each transition value and the time at which it occurs as a list of `TN VAL` value pairs. In its second form, all consecutive signal values are listed in order after the **`PATTERN`** keyword, and a hold time specifies the duration of each signal value specified.

Bus values for both the standard and pattern definitions accept exactly the same syntax.

- parameters are denoted $(P) or 'P'
- values can be specified with or without double quotes and use the notation HX[...], DX[...], OX[...], and BX[...] to specify in which base this number is given (by default it is the base of the bus). All these notations can be mixed in the same bus definition, for example:

```
.sigbus a[0:7] VHI=3 VLO=0 THOLD=10ns BASE=BIN
+ PATTERN $(PAT_RAMP) 101 "001" 'PAT2' BX010 "DX123"
```

Parameters can only be strings, except for buses declared with BASE=DEC. This is mandatory to allow sweeps. If this rule is not satisfied, the following message is issued:

```
ERROR 1545: COMMAND .SIGBUS: bus A, parameters of type real can only be
used with buses using base=DEC
```

If the parameter holds a decimal value, the message issued is:

```
ERROR 1544: COMMAND .SIGBUS: decimal non-integer value found in bus A
```

The sweeping of literal (string) parameters is allowed if the parameters do not effect anything but the **`.SIGBUS`** command. The other uses of such parameters are unchanged. A literal or string parameter is a parameter declared with the value enclosed in double quotes.

Parameters

- BNAME

  Bus name, previously defined via the **.SETBUS** command, or can be implicitly declared with MSB:LSB syntax specified. A limitation of the simplified BNAMEMSB:LSB notation is that bus names cannot contain numbers: BUS18:0 will be equivalent to BUS[18:0] but not BUS1[8:0].

- MSB:LSB

  Series of bit names, the most significant bit being defined first. This mechanism can be used to implicitly declare **.SETBUS**.

- TN

  Time in seconds at which the bus signal is equal to VAL Volts.

- VAL

  Bus signal voltage level at time TN.

- **VHI**=VAL

  Upper bus signal voltage level. Default is 5V.

- **VLO**=VAL

  Lower bus signal voltage level. Default is 0V.

- **TFALL**=VAL

  The time for a falling signal to reach **VLO** Volts. Default is 2ns.

- **TRISE**=VAL

  The time for a rising signal to reach **VHI** Volts. Default is 2ns.

- **THOLD**=VAL

  The hold time for each signal value, measured from the 50% point of the transition.

- **TDELAY**=VAL

  The delay time before a signal pattern starts.

- **BASE**

  Keyword indicating that the bus signal number system is to be defined.

  OCT

  Keyword indicating that bus signals are defined in octal.

  DEC

  Keyword indicating that bus signals are defined in decimal. Default.

  BIN

  Keyword indicating that bus signals are defined in binary.

HEX

Keyword indicating that bus signals are defined in hexadecimal.

- **P**

  Keyword indicating the bus signal is periodic.

- **SIGNED**=**NONE**|**1COMP**|**2COMP**

  Defines how negative values inside bus patterns are managed.

  NONE—negative values are forbidden (default)

  1COMP—the one's complement of the value is done

  2COMP—the two's complement of the value is done

  Must be specified before the **PATTERN** parameter when specified together.

- **PATTERN**

  Keyword indicating the bus signal is pulsating (digital-like). Integer values can be specified, a bus value specifying `base=bin pattern 101` has the same effect as specifying `base=dec pattern 5`. The **PATTERN** on a **.SIGBUS** command can also be specified via a parameter using `$(PAT)` where `PAT` is the parameter as specified in the **.PARAM** statement. For example:

```
.PARAM PAT=3
.SIGBUS toto VHI=1.8 VLO=0 TFALL=100ps
+ TRISE=100ps THOLD=5ns TDELAY=0 BASE=DEC
+ PATTERN $(PAT)
```

- `$(PAT)`

  `PAT` is a previously declared parameter in the special case of the **PATTERN** specification above.

- **Z**

  Releases the applied signal of a bus. When the Z state is active, the signal will be disconnected from the node, and the node will be computed by Eldo as if it were a non-input signal.

- **FILE**=file

  Allows Eldo to read values from the specified *file*. This is a text file that supplies the time-digital value pairs. The file can have multiple lines and can have any number of point pairs per line. Parentheses are not allowed in this file, and continuation line signs "+" are optional. The format of the input file is the same as that used when specifying an input file for a PWL source, see the "Piece Wise Linear Function" on page 338.

Examples

.SIGBUS can be used to implicitly declare **.SETBUS**, providing that `BNAMEMSB:LSB`, `BNAME[MSB:LSB]` or `BNAME<MSB:LSB>` syntax is used as shown below:

```
.SIGBUS BUS2:0
.SIGBUS SELECT[2:0]
.SIGBUS DOUT<3:0>
```

is equivalent to:

```
.SETBUS BUS BUS2 BUS1 BUS0
.SIGBUS BUS
.SETBUS SELECT SELECT[2] SELECT[1] SELECT[0]
.SIGBUS SELECT
.SETBUS DOUT DOUT<3> DOUT<2> DOUT<1> DOUT<0>
.SIGBUS DOUT
```

A “Z” state can be specified for a single signal of a bus, shown in the example below:

```
.SIGBUS B<1:0> VHI=2 VLO=0 TFALL=1n TRISE=1n THOLD=200n
+ TDELAY=0 BASE=BIN
+ PATTERN 00 01 0Z 00 01
```

The example below shows a **.SIGBUS** command with file specification:

```
.SIGBUS A[3:0] VHI=1 VLO=0 THOLD=1  BASE=BIN FILE=bus_input
```

This example uses the following file, *bus_input*, as input:

```
---------------- bus_input file -------------------------
# Testing .SIGBUS A[3:0]
0 1111
10e-9 0001
20e-9 0010
30e-9 0100
40e-9 1000
```

The example below shows the sweeping of literal (string) parameters is allowed, but only when used in conjunction with a **.SIGBUS** command. The example declares a bus and assigns it using hexadecimal value HX000. Then the default value is replaced by “HX001”, a transient simulation is performed, and the value is replaced by “HX010”.

```
.param pat="HX000"
.sigbus vplus[9:0]
+ VHI=vdd
+ VLO=0
+ BASE=DEC
+ PATTERN $(pat)

.step param pat list "HX001" "HX010"
```

.SINUS

Sinusoidal Voltage Source

```
.SINUS NODE VO VA FR [TD [THETA]]
```

Parameters

- `NODE`

  Node name between which the source is connected to and ground.

- `VO`

  Offset voltage in volts.

- `VA`

  Sine wave amplitude in volts.

- `FR`

  Frequency in Hertz.

- `TD`

  Delay time in seconds. Optional.

- `THETA`

  Damping factor in s^{-1}. Optional.

Note

The generated waveform is described by:

$$V = VO + VA \times \sin(2\pi \times FR(t - TD)) \times exp(-(t - TD) \times THETA)$$

For more information, refer to the Sources chapter.

Example

```
.sinus n2 4.0 1.0 1meg 0.0
```

Specifies a sine wave applied between node `n2` and ground with a 4V offset voltage, 1V amplitude, 1MHz frequency and zero delay.

.SNF

Spot Noise Figure

```
.SNF INPUT=(LIST_OF_DEVICES) OUTPUT=(LIST_OF_DEVICES)
+ [INPUT_TEMP=VAL] [NOISETEMP=VAL]
```

A new number, SNF (Spot Noise Figure), is calculated. When specifying the **INPUT_TEMP**=VAL, the noise at the input will be computed at **INPUT_TEMP**. When **NOISETEMP** is specified in the **.SNF** command, this is equivalent to specifying the parameter on all sources in the netlist. This option will take priority over the **INPUT_TEMP** parameter whether it is used on a source or in the **.SNF** command. Then, the total noise contribution will be calculated by:

$$SNF = \frac{\text{(Input noise at INPUT\_TEMP+Other noise sources at TREF - Output noise at TREF)}}{\text{Input noise at INPUT\_TEMP}}$$

Note

When **NOISETEMP** is specified, it will replace **INPUT_TEMP** in the equation above.

If neither **INPUT_TEMP** nor **NOISETEMP** are not specified, then the method of calculation will be:

$$SNF = \frac{\text{(Noise in circuit - noise due to output)}}{\text{Noise due to source}}$$

Multiple **.SNF** commands are supported in Eldo and Eldo RF.

Parameters

- LIST_OF_DEVICES

  This can contain wildcard '*' characters anywhere in the name. Each item must be separated with a white space or a comma. Brackets are optional. However, if more than one name is provided, then commas and brackets *must* be used.

Example

```
.SNF INPUT=(XM1*.M*,XM[1-3]*) OUTPUT=M19
```

The example above shows how it is possible to use wildcard characters.

```
.SNF INPUT=(XM1,XM2*) OUTPUT=M19
```

This number is frequency dependent, hence a curve is generated that can be plotted via **.PLOT NOISE SNF** in exactly the same way one can plot:

```
.PLOT NOISE INOISE
```

Note

The **`.SNF`** command provides results only if a **`.NOISE`** or **`.SSTNOISE`** (Eldo RF) command is present.

.SOLVE

Sizing Facility

```
.SOLVE PARAM param_name MIN MAX expr=expr [TOL=VAL]
+ [RELTOL=VAL] [GRID=VAL]
.SOLVE obj_name [W|L] MIN MAX expr=expr [TOL=VAL]
+ [RELTOL=VAL] [GRID=VAL]
.SOLVE CNAME [W|L] MIN MAX OPSIZE [TOL=VAL]
```

Used in conjunction with a DC analysis only. Eldo sizes the specified component to match given constraints. This command can be used to solve the value of a parameter. The target may be `V(NODE)=<VAL>`, or `I(Vxx)=<VAL>`, or an expression (for example `V(NODE1)+V(NODE2)`).

Parameters

- `CNAME`

  Name of the component to size. Legal names are **R**, **MOS**, **V**xx, **I**xx.

- `MIN, MAX`

  Bounds to search for a solution, expressed in correct units.

- `OPSIZE`

  Request component sizing so that the voltage on a specific node or current through a voltage source matches given constraints. The syntax is as follows:

  `V(NODE)=VAL`
  Specifies that the component should be sized so that the voltage on the specified node be equal to `VAL`, with tolerance **TOL**.

  `I(Vxx)=VAL`
  Specifies that the component should be sized so that the current through the specified voltage source be equal to `VAL`, with tolerance **TOL**.

- **W**, **L**

  Keywords identifying width or length for MOS transistors. Optional.

- **TOL**=VAL

  Required accuracy, specified in absolute units. Default is the value of the **EPS** parameter. Optional.

- **GRID**=VAL

  Means the result must be a multiple of `VAL`. Optional.

- **RELTOL**=VAL

  Adds the relative tolerances. Optional.

There are two points to bear in mind when using the **.solve** command:

- **.solve** works in the case where there is a 0 in the interval, and if the function is monotonous in the interval, but cannot be used to find a MIN or a MAX.

- the default absolute tolerance for **.solve** is **EPS**; specify another value in the **.solve** command to reset this default.

Examples

```
.solve m1 w 10u 50u v(2)=0.245v tol=1u
```

Specifies that the width of transistor `m1` be sized to make the voltage on node `2` equal to 0.245V with a tolerance of 1µV. Maximum and minimum values of the width are 50µm and 10µm respectively.

```
.solve m3 l 1.5u 3.5u i(v2)=0.2m tol=0.05m
```

Specifies that the length of transistor `m3` be sized to make the current through voltage source `v2` equal to 0.2mA, within a tolerance of 0.05mA. Maximum and minimum values of length are 3.5µm and 1.5µm respectively.

.START_TIME

Simulation Start Time

```
.START_TIME time_value
```

Forces Eldo to start the simulation at a specified time (`time_value`) instead of at time 0. DC input signals are evaluated at time `time_value`, not at time 0. This command is similar to the **`.ADMS_START`** command available in Questa ADMS. It can be useful if you already have stimuli that you want to reuse instead of rewriting it.

Parameter

- `time_value`

  Simulation start time. Default is 0.

Example

```
V1 1 0 pwl 0 0 10n 10 15n 10
.start_time 10n
```

The DCOP value for node 1 will be 10 V, because that is the value at the 10ns specified simulation start time.

.STEP

Parameter Sweep

```
.STEP TEMP|DIPOLE INCR_SPEC
.STEP MOS W|L INCR_SPEC
.STEP MNAME PARAM_NAME INCR_SPEC
.STEP PARAM PARAM_NAME INCR_SPEC, {[VALSTART] VALSTOP VALUE}
.STEP ITEM INCR_SPEC {ITEM2 BOUND}
.STEP (ITEM1,ITEM2... ITEMn)
+ LIST | = (VALi1, VALi2... VALin)... (VALj1, VALj2... VALjn)
.STEP (ITEM1,ITEM2... ITEMn) LIST FILE=FILE
```

Used to perform several simulations while sweeping one circuit parameter or several circuit parameters simultaneously (multiple-sweep). Nested sweeps can also be specified. Further nesting levels can be applied by additional **.STEP** commands. There is no limit to the depth (levels) of nested sweeps possible.

By default, the second run in **.STEP** uses the result of the previous STEP as an initial guess. Setting option **NOMEMSTP** means Eldo will not use the results of the previous STEP run as an initial guess for the next one.

When specifying multiple sweeps, secondary `incr_spec`'s cannot define the number of points. Eldo will determine the number of points from the first `incr_spec` defined and then set the same number for subsequent increment steps. This makes it possible to set multiple items and have them change concurrently, then subsequent **.STEP** commands would provide the next level of nesting.

The series of parameters on the right of the equals sign indicate a series of 'n' values enclosed in brackets, which must be specified. The number of terms of this series corresponds to the number of simulations, and 'n' items would change at each simulation.

To perform a parameter vector sweep, with all parameters taking their values from lists, use the last syntax shown above and ensure that `ITEM` is correctly specified for parameter items, for example specifying `(P(IG1), P(LR1))` to sweep parameters IG1 and LR1.

The command has the ability to sweep several parameters at the same time using the **.STEP PARAM** syntax. Multiple increment step specifications can be made. This is in order to be able to have "windows" with more points than in other regions.

The sweeping of literal parameters (parameters declared with the value enclosed in double quotes) is only allowed when used in conjunction with a **.SIGBUS** command.

.MPRUN can be used to take advantage of multi-processor machines for the **.STEP** command. **.ALTER** and **.TEMP** have higher priority than **.STEP** when used with **.MPRUN**. Please see ".MPRUN" on page 729 for further information.

Parameters

- **TEMP**

  Keyword indicating that temperature is to be swept.

- DIPOLE

  Name of the Dipole (R, C, L, V, I) component whose value is to be swept.

- MOS

  Name of the MOS component whose **W** or **L** parameter is to be swept.

- **W**, **L**

  Keywords identifying either MOS width or length respectively.

- MNAME

  Model name, of which a parameter PARAM_NAME is to be swept.

- PARAM_NAME

  Name of the parameter to be swept.

- **PARAM**

  Keyword to identify that a globally declared parameter is to be swept. Multiple increment step specifications can be made. Non-primitive parameters can be specified.

- **LIST**

  Keyword specifying that a list of individual values are to be swept.

Note

For multiple increment step specifications: Be careful of the character "," which is used for separating the different windows. If not specified, VALSTART is assumed to be the VALSTOP value of the preceding window.

Note

.TRAN ... **SWEEP** will not work if **PARAM** is specified and a warning message will be issued.

- ITEM

  Can be one of the following:

  P(global_var)
  E(device,parameter)
  M(model_name,parameter)
  EM(device_name,model_parameter_name)
  LIB([**KEY**=]libname)

  P

  Parameter item.

E
Element item.

M
Model item.

EM
Model item attached to a specific Element. Affecting the model parameter of this new private model created for `device_name`, leaves the original model unchanged. Once **`.STEP`** is complete, this private model is discarded, and the original model is attached back to the device.

LIB
Library item. A list of strings is expected, of type `LIST` as specified in `INCR_SPEC` below. Used to make several Eldo runs, each of them using a different variant of the libraries. The library can be identified using its key instead of using its filename (with full library path). In this case the **`KEY`** keyword is mandatory. If not set, Eldo will consider the argument of the **`.STEP LIB`** as a filename.

- `INCR_SPEC`

  Specifies the increment step of sweep.

  Specification of the increment step may be either of the following:

```
VALSTART VALSTOP [DEC|OCT|LIN|INCR] VALUE
LIST {VAL1 VAL2 ... VALN}
```

`VALSTART`
Initial value of sweep.

`VALSTOP`
Final value of sweep.

LIN
Optional keyword, explicitly selecting a linear sweep.

INCR
Incrementing value of sweep. This is the default.

DEC
Keyword to select a logarithmic sweep.

OCT
Keyword to select an octave sweep.

LIST
Keyword specifying that a list of individual values are to be swept. Accepts character strings for dealing with the `ITEM` of type **`LIB`**`(libname)`.

`VALUE`
Specifies the number of points per decade and octave for DEC and OCT respectively, the total number of points for LIN, and the value of increment for INCR.

`VAL1..VALN`
A list of values to be applied to the specified parameter at each point in the sweep.

- `BOUND`

  Same as `INCR_SPEC`, except that only boundaries are provided. The increment value is chosen according to the number of runs imposed by the first swept parameter.

- **`FILE`**`=file`

  Allows Eldo to read values from the specified *file*. This is a text file that supplies the time-current (tn in) or time-voltage (tn vn) pairs. Engineering units (for example 9ns) are allowed. The time-value pairs are separated by spaces, commas or newline characters. The file can have multiple lines and can have any number of point pairs per line. Parentheses are not allowed in this file, and continuation line signs "+" are optional.

Note

To use this command with a device that is situated in a subcircuit, use the nested output format.

For more details, refer to ".PRINT" on page 830.

Parameters **`TEMP`** or **`PARAM`** specified in the **`.STEP`** command may also appear on the x-axis of a graphical output when performing a DC analysis.

Eldo will select the model to be assigned to MOS devices according to the geometric size of each device, even if these geometric sizes are modified at run-time via **`.STEP`** commands. In previous versions, the selection of the model was done just once at the very beginning of the simulation, and was not changed at run time.

Examples

```
.step temp 25 35 2
```

Specifies that simulator runs should be carried out between the temperatures 25 and 35 degrees in increments of 2 degrees Celsius.

Note

The **`INCR`** keyword is optional as this is the default.

```
.step mos_1 w 25u 40u 5u
```

Specifies that simulator runs should be carried out with the width of the transistor `mos_1` being swept from 25µm to 40µm in increments of 5µm.

```
.step qmod rb 80 110 10
```

Specifies that simulator runs should be carried out with the base resistance parameter of devices with transistor model `qmod` being swept from 80Ω to 110Ω in increments of 10Ω.

```
c1 1 2 20p
...
.step c1 1p 10p 1p
```

Specifies that simulator runs should be carried out with the capacitor `c1` being swept from 1pF to 10pF in steps of 1pF.

```
.step x1.r1 0.1k 0.5k 0.1k
```

Specifies that simulator runs should be carried out with the value of the resistor `r1`, instantiated using the subcircuit instance name `x1`. The resistor value should be swept from 0.1kΩ to 0.5kΩ in increments of 0.1kΩ.

```
.param r1 1k
...
.step param r1 1k 2k 1k, 2k 4k 500
```

Specifies that the value of resistor r1 will be swept from 1kΩ to 2kΩ with a step of 1kΩ, and then from 2kΩ to 4kΩ with a step of 500Ω.

```
.param p2=100k
.param p1=sqrt(p2)
...
.step param p1 1k 5k 1k
```

This example shows how a non-primitive parameter `p1` can be specified.

The following example shows how the command can be used to define a multiple run by a list of parameters:

```
.step param (p1 p2 ...pn) list (val11 val21 ...valn1)
+ ...(valj1...valjn)
```

This will perform `j` runs:

```
run1: p1=val11, p2=val21 ... pn=valn1
run2: p1=val21 ...
...
runj: p1=valj1, ... pn=valjn
```

The following example declares a bus and assigns it using hexadecimal value HX000. The default value is then replaced by “HX001”, a transient simulation is performed, and the value is replaced by “HX010”.

```
.param pat="HX000"
.sigbus vplus[9:0]
+ VHI=vdd
+ VLO=0
+ BASE=DEC
+ PATTERN $(pat)

.step param pat list "HX001" "HX010"
```

The following example shows how the command can be used to sweep Models attached to specific Elements:

```
M1 ... NMOS W=...
M2 ... NMOS W=...
.MODEL NMOS NMOS VT0 = 1

.STEP EM(M1,VT0) 0 5 1
```

For this example, Eldo will:

1. Emulate a command **.MODDUP** M1
   A new model NMOS.M1 is created, which is private to M1.
   All parameters of NMOS.M1 are initialized to the NMOS values.
2. Five Simulations will be performed, for NMOS.M1.VT0 varying from 0 to 5. NMOS.VT0 remains at 1.0.
3. NMOS.M1 is discarded: model attached to M1 is NMOS again.

Note

As in the **.MODDUP** case, parameter dependencies are propagated.

```
.PARAM P1 = 550
M1 ... NMOS W=...
M2 ... NMOS W=...
.MODEL NMOS NMOS VT0=1 UO=P1

.STEP (EM(M1,VT0),P(P1)) = (1,550) (2,600)
```

In the example above, for both NMOS.M1 and NMOS, the UO value will be identical.

The following examples show how the command can be used to sweep Parameters, Elements, Models, or Models attached to specific Elements:

```
.PARAM P1=1u
.MODEL NMOS NMOS LEVEL=1 VTO=2
M1 p1 p2 p3 p4 NMOS w=p1 l=3u
M2 p1 p2 p3 p4 NMOS w=p1 l=3u
...
.end
```

After the above specification, the following four **.STEP** commands are valid:

```
.STEP P(P1) 1u 10u 1u
```

Parameter P1 varies from 1µ to 10µ in steps of 1µ.

```
.STEP E(M1,l) 3u 4u 1u
```

Length of Element M1 varies from 3µm to 4µm in steps of 1µm.

```
.STEP M(NMOS,VT0) 1 3 1
```

Parameter VT0 of model NMOS varies from 1 to 3 in steps of 1.

```
.STEP EM(M2,VT0) 1 4 1
```

A private model will be assigned to `M2`. `VT0` of model attached to `M2` will vary from 1 to 4 in steps of 1; `VTO` of model attached to `M1` will remain unchanged.

The following defines a two-level nested sweep. The first loop is defined by the first **`.step`**. The resistor, `R22`, will be swept from 20kΩ to 100kΩ in a linear 1kΩ increment. This defines 81 points for the first loop.

```
.STEP E(R22,R) 20K 100K LIN 1K M(MOS1,COX) 0.4M 0.3M
.STEP P(GLOBAL_VAR) 1 1000 DEC 10 E(C_CUPL,C) 100p 500p LIN
```

The second item, the MOS model named `MOS1` would have the parameter `COX` swept from 0.4M down to 0.3M in increments that produce 81 points (that is, *abs(0.3M-0.4M)/81*). The order is important and would need to be preserved. The number of points is defined by the first increment specification so that the entire **`.step`** command will have a consistent number of points.

The resistor and `COX` parameter would increment/decrement together:

```
(20K, 0.4M)
(21K, 0.38977M)
.
.
(100K, 0.3M)
```

The second nesting level is defined by the second **`.step`** command. The global parameter named `GLOBAL_VAR` will be swept from 1 to 1000 in decade logarithmic steps, producing 31 increments. The capacitor, `C_CUPL`, would be stepped from 100pF to 500pF linearly with 31 steps together with the `GLOBAL_VAR` parameter.

```
.STEP PARAM (E(R1,R) TEMP LIB(corna.lib)) LIST
+ (20k 27 typ)
+ (10k -40 fast)
```

Specifying the above means that two simulations will be performed:

- The first with `R1` value of 20k, `TEMP` is 27 and uses the `LIB` variant `typ` for file `corna.lib`.
- The second with `R1` value set to 10k, a `TEMP` value of –40 and the `LIB` variant `fast` of the file `corna.lib`.

The following example shows a parameter vector sweep, with all parameters taking their values from lists. Eldo will sweep parameters IG1 and LR1 over four simulations. The first simulation with IG1 at 10u and LR1 at 1000u. The last simulation with IG1 at 100u and LR1 at 100u.

```
.param IG1=10u LR1=1000u
.step (p(IG1),p(LR1)) LIST (10u, 1000u) (20u, 500u) (50u, 200u)
+ (100u, 100u)
i1 1 0 ig1
r1 1 0 '10e6*lr1'
.plot dc v(1)
.dc
```

The following example shows how the **.STEP** command is used to make several Eldo runs, each of them using a different variant of the libraries, using the keyword `LIB(libname)`:

```
.LIB mylib TYP
...
.STEP LIB(mylib) TYP MIN MAX
.END
```

Three simulations will be performed: one using the variant `TYP` of `mylib`, another one using the variant `MIN`, and a last one using the variant `MAX`.

Note

The `LIB` specification can be used with any other specification.

Note

`TYP`, `MIN`, and `MAX` are not keywords, but are the character strings that can appear as the 2nd arguments of the **.LIB** command (**.LIB** `FNAME [LIBTYPE]`). Usually, variant names are `TYP`, `MIN`, and `MAX`, but they can be any string.

```
.STEP P(P1) 1 2 0.5 LIB(mylib) TYP MIN MAX
```

Here, Eldo will perform three simulations:

- one with both `P1`=1.0 and `mylib` using variant `TYP`
- one with both `P1`=1.5 and `mylib` using variant `MIN`
- one with both `P1`=2.0 and `mylib` using variant `MAX`

Limitations on Library Variants

- Subcircuits are not replaced.
  When **.LIB** is specified in the input file, it can be used to select a subcircuit to be included in the design. Taking a subcircuit from another library than the library specified in the input file could result in a change of topology (and changing topology of the current design is strictly impossible), therefore Eldo would issue an error whenever the user attempted to substitute one subcircuit for another one.
- Switching between GUDM and non-GUDM models is prohibited.
  Similarly, switching between GUDM models is prohibited.
- Only MOS, BJT, DIODE, JFET, R, L, and C **.model** commands can be substituted.
  Attempting to substitute other kinds of models will lead to an error.

.SUBCKT

Subcircuit Definition

```
.SUBCKT NAME NN {NN} [(ANALOG|OSR|DIGITAL)]
+ [(NONOISE)] [(INLINE)] [[PARAM:] PAR=VAL {PAR=VAL}] [STATISTICAL=0|1]
...
<CIRCUIT_COMPONENTS>
...
.ENDS [NAME]
.SUBCKT LIB FNAME SNAME [LIBTYPE]
```

A subcircuit consists of Eldo, FAS and FIDEL elements. The subcircuit is defined in the circuit description file by groups of elements beginning with a **.SUBCKT** command and terminating with a **.ENDS** command. All components and nodes used in the subcircuit definition are known only locally, unless they have been globally defined via the **.GLOBAL** or **.PARAM** commands. **.MODEL** specifications inside the subcircuit are also known only locally. There is no limit to the size or complexity of the subcircuit.

Note

Care must be taken when using the **.PARAM** command to ensure that no unwanted parameter assignments are made.

Subcircuit definitions may be nested. The inner subcircuit definitions are local to the subcircuit definition in which they are defined, that is, they are not valid outside of it.

Subcircuits may be parameterized. Symbols may be used as values within the subcircuit body which may be assigned values when the subcircuit is instantiated. A subcircuit may be stored in a library file, in which case the second syntax above and the **.LIB** or **.ADDLIB** commands should be used. The second syntax allows no continuation lines.

The subcircuit may optionally be simulated using the differentiated accuracy system. The iteration technique used to process the subcircuit is chosen by the **(ANALOG)** optional keyword. Usually, a circuit with **(ANALOG)** specified indicates ‘high accuracy’.

Note

.MACRO is equivalent to **.SUBCKT**.

By default, when Eldo encounters multiple definitions of **.SUBCKT** statements an error is reported and the simulation is stopped. Specify option **USEFIRSTDEF** to force Eldo to only use the first definition (any further definitions are ignored) to allow the simulation to proceed.

Parameters

- NAME

  Name of the subcircuit. May also be specified with the **.ENDS** command if desired.

- `NN`

  Names of the subcircuits nodes. Nodes are referenced in the same order that they are called in the subcircuit call statement. Ground (or 0 V) may not be referenced in this list.

- **`LIB`**

  Keyword indicating a subcircuit library file is to be used.

- `FNAME`

  Name of the library file that contains the subcircuit description.

- `SNAME`

  Name of the subcircuit stored in library file `FNAME`. See “Library variant management” on page 694 in the **`.LIB`** command description for further details.

- **`(ANALOG)`**

  Keyword used with the differentiated accuracy system indicating that the subcircuit should be solved using Newton block iteration techniques. These techniques are used in conjunction with the **`EPS`** parameter in the **`.OPTION`** command. **`(ANALOG)`** basically means “high accuracy.”

> Before using the differentiated accuracy system see the relevant section in the Speed and Accuracy chapter.

- **`(OSR|DIGITAL)`**

  Used to stop propagation of the **`ANALOG`** flag across the hierarchy. In addition the flag will request Eldo to use OSR in the selected blocks, if possible (that is, MOS subcircuit with **`OSR`** flag could then be solved by OSR, but BJT subcircuit will still be solved by Newton even if flag **`OSR`** is set).

- **`(NONOISE)`**

  Specifies that all subcircuit objects are noiseless. However, the appearance of a flag **`NOISE`** at a lower level of subcircuit hierarchy overrides the effect of a **`(NONOISE)`** flag at a higher level. This parameter must be specified before any **`PARAM:`** specification, if any.

- **`(INLINE)`**

  When specified, Eldo will not print the full hierarchical name of the device which has the same name as the **`.SUBCKT`**. The effect of this option is for print out in the **`.OP`** table only. This parameter must be specified before any **`PARAM:`** specification, if any.

- **`PARAM:`**

  Keyword indicating parameter allocation within the subcircuit definition. Optional.

- `PAR=VAL`

  Specifies that the parameter `PAR` is assigned the value `VAL` inside the subcircuit, unless another value is assigned to the parameter when the subcircuit is instantiated.

In case of M factor usage, see also "M53" on page 979. With some descriptions, Eldo will generate the following error message:
`ERROR 712:SUBCKT "XXX": cannot define a parameter named 'M'`

- **STATISTICAL**=0|1

  Specify whether any statistical variation due to Monte Carlo, worst case, or DC mismatch analysis can be applied to the subcircuit. 0 means the subcircuit components will keep their nominal values. 1 means the subcircuit components have statistical variation applied. The global default can be specified via option **STATISTICAL**. Default is 1.

- NAME

  Name of the subcircuit. May or may not be specified.

- LIBTYPE

  Name of a library variant to be used.

For details of subcircuit instance syntax refer to "Device Models" on page 119.

Related options

USEFIRSTDEF (see "USEFIRSTDEF" on page 962), **DSCGLOB**, see "DSCGLOB=X|GLOBAL" on page 976

Examples

```
.subckt inv n1 n2 n3 n4 param:p1=10u r1=8u
m1 n3 n2 n4 n4 pmos w=p1 l=r1
m2 n3 n2 n1 n1 nmos w=p1 l=r1
.ends inv
...
x1 vss n8 n9 vdd inv p1=5u r1=5u
```

Specifies the subcircuit `inv`. Nodes declared in the subcircuit are local. The parameters `p1` and `r1` are assigned values both at instantiation and in the **.SUBCKT** command.

```
.subckt w1 n1 n2 n3 n4 (analog)
+ param:r1=10u r2=5u
m1 n1 n4 n5 0 nmos w=r1 l=r2
m2 n5 n5 n2 0 nmos w=r1 l=r2
m3 n4 n5 n3 0 nmos w=r1 l=r2
.ends w1
...
x1 vss 0 0 vdd w1
```

Specifies the subcircuit `w1`. This subcircuit is defined to be solved only using Newton block iteration techniques due to the presence of the **(ANALOG)** optional keyword.

```
*SUBCKT definition
.subckt inv n1 n2 n3 n4
```

```
m1 n3 n2 n4 n4 pmos w=w1 l=l1
m2 n3 n2 n1 n1 nmos w=w2 l=l2
.param l1=5u
.ends inv
...
*main circuit
x1 vss n8 n9 vdd inv w1=60u w2=20u
.param l2=5u
```

Specifies the subcircuit `inv` using variables `w1`, `l1`, `w2` and `l2`. Note the use of the **.PARAM** command in this example. The value of the parameter `l1` has been assigned a value in the subcircuit `inv` and so is local to this subcircuit, whereas the parameter `l2` has been assigned a value on the main circuit level and so any subsequent instances of the parameter `l2` in other subcircuits would also be assigned this value.

```
.subckt outer n1 n2 n3 n4
q1 n1 n5 n6 pbip
...
   .subckt inner n1 n2 n3
   r1 n1 n3 1k
   r2 n3 n8 4k
   ...
   .ends inner
x1 n10 n12 vdd inner
...
.ends outer
x1 n21 n25 n30 vdd outer
```

Specifies and instantiates a subcircuit `inner` nested inside another subcircuit `outer`.

Order of Precedence of Subcircuit Parameter Assignments

Parameters used within a subcircuit may be defined in several ways.

- In the subcircuit instantiation:

```
X1 N1 N2 N3 MY_SUBCIRCUIT P3=10
```

- Internally, within the subcircuit:

```
.SUBCKT MY_SUBCIRCUIT
.PARAM p3=20
.ENDS
```

- In the subcircuit declaration:

```
.SUBCKT MY_SUBCIRCUIT P1 P2 P3 PARAM: p3=2
```

- Or globally in the top level netlist:

```
.PARAM p3=5
X1 N1 N2 N3 MY_SUBCIRCUIT
```

The precedence of these parameter assignment methods is, in descending order, subcircuit instantiation, internal subcircuit, subcircuit declaration which acts as a default only, and finally global declaration.

Use option **PARHIER** to control the priorities for parameters.

Note

.PARAM assignments are order dependent within the netlist. Thus, a parameter value assigned using the **.PARAM** command will only apply to subcircuits instantiated after this **.PARAM** command. Therefore, for the below:

```
.SUBCKT INV
.PARAM p1=v0
.ENDS
x1 ... inv
.param p1=v1
```

This is accepted, and `p1` retains the value of `v0` within `x1` (when option **PARHIER** is set to local).

Accessing Nodes Inside Subcircuit Instances from Outside

The following example illustrates how inner nodes may be accessed from outside of a subcircuit instance in which they defined.

```
.subckt inner n1 n2 n3
r1 n1 n3 1k
r2 n3 n8 4k
q27 n1 n7 n8 nbip
...
.ends inner
.subckt outer n1 n2 n3 n4
q1 n1 n5 n6 pbip
...
x1 n10 n12 vdd inner
...
.ends outer
x13 n21 n25 n30 vdd outer
cmill1 x13.n1 x13.n5 1.2p
cmill2 x13.x1.n1 x13.x1.n5 0.9p
```

Where `cmill1` is the Miller Capacitance of `q1` located inside the subcircuit `x13` and `cmill2` is the Miller Capacitance of `q27` located inside `x1` which in turn is located inside `x13`. Note that in this example, the capacitances are declared from outside of the subcircuit definitions.

(INLINE) Keyword Example

The following example illustrates how the **(INLINE)** keyword can be specified on the **.SUBCKT** instance, so that Eldo will not print the full hierarchical name of the device which has the same name as the **.SUBCKT**.

```
X1 ... MFOO
X2 ... MFOO
.SUBCKT MFOO ... (INLINE)
M1 ...
MFOO
M2
...
```

```
.ENDS
```

Then, in the OP table, it will not be `X1.MFOO` and `X2.MFOO` which appear, but simply `X1` and `X2`.

`X1.M1`, `X2.M1`, `X1.M2` and `X2.M2` names will be printed out unaffected.

DSCGLOB Option Example

Nodes which appear in subcircuit definitions have priority over the nodes defined with **.GLOBAL** statements. Option **DSCGLOB** can be used to change this behavior. For example:

```
.GLOBAL QWE
VG QWE 0 DC 3
*.SUBCKT with a global node name in argument list
.SUBCKT B QWE
R1 QWE 0 1K
.ENDS
V2 PIN2 0 DC 2
X2 PIN2 B
```

By default, Eldo will connect R1 between PIN2 and 0 because the symbolic subckt pin QWE is related to real node PIN2.

When option **DSCGLOB=GLOBAL** is specified Eldo will connect R1 between global node QWE and 0.

.SUBDUP

Subcircuit Duplicate Parameters

```
.SUBDUP SUBCKT_INST_NAME
```

This command is used to inform Eldo of the duplicate parameters and models which are local to the subcircuit. Also, it allows access to parameters in the hierarchical form for SimPilot. This command is useful when Eldo is running in conjunction with SimPilot.

Example

```
.subckt xa 1 2
.param p1 = 2
.param p3 = p1*p2
r1 1 2 p3
.ends xa

i1 1 0 1
x1 1 0 xa p2 = 1
i2 2 0 1
x2 2 0 xa p2 = 1
.dc

*.step param xa.p1 5 6  1  ! step p1 in both instance X1 and X2

.subdup x1
.step param x1.p1 5 6 1  ! step p1 only in instance X1
```

In the above, the first **.STEP** command, which is commented out, would step the parameter p1 of both instance X1 and X2, as defined in the common subcircuit XA. This is not as required. In order to change the parameter p1 only in X1, first a **.SUBDUP** command has to be specified, so that x1 will have a private copy of parameter p1, which can then be swept. The final **.STEP** command will step parameter p1 only in instance X1.

.TABLE

Value Tables

```
.TABLE NAME (X1 Y1) {(XN YN)}
```

This command defines tables of run time values to be used in AC, DC or Transient analyses. Values from the table may then be accessed from arithmetic expressions using the function call **TABLE(**NAME**)**.

Parameters

- NAME

  Name of the table being defined.

- Xxx Yxx

  Data value pairs, whose units depend on the analysis type.

Note

The analysis specification (AC, DC, TRAN) is deprecated. It will not print an error if specified but it is not used by Eldo.

Example

```
.table ac_table_ex ac (1 1) (10k 10)
.defwave wave1=vdb(s)-table(ac_table_ex)
```

Here, ac_table_ex contains the AC analysis values 1 at 1 Hz and 10 at 10 kHz.

.TCL_WAVE

Waveform Definition Using a Tcl Function

```
.TCL_WAVE [ANALYSIS] WAVE_NAME=TCL_FUNCTION_CALL
```

Used to define a new waveform using a Tcl function. It is similar to the **`.DEFWAVE`** command with restrictions applied on the waveform expression, which must be a single Tcl function. The resulting waveform can be used as a normal **`.DEFWAVE`** with syntax `W(WAVE_NAME)`.

This allows you to dynamically manage User Defined Functions (UDF) written in Tcl language. The functions of the post-processor library and other commands are available through the Tcl interface.

For loading a Tcl file containing functions into the Eldo Tcl interpreter, see the command ".USE_TCL" on page 929.
For performing a single call to a Tcl function, see the command ".CALL_TCL" on page 559.
For further information on Tcl commands, see the Post-Processing Library chapter of this manual.

Parameters

- **`ANALYSIS`**

  Type of analysis to be used.

- **`WAVE_NAME`**

  New waveform name.

- `TCL_FUNCTION_CALL`

  The Tcl function. A function can take any kind of arguments: waves, numbers and keywords. These keywords are: IRUN, ICARLO, IALTER which respectively represent the index of the current step, Monte Carlo run, and **`.ALTER`**. NBRUN, NBCARLO, NBALTER which represent the number of steps, Monte Carlo runs, and .ALTER.

Example

```
.extract tran label=minin min(v(in),10n,260n)
.extract tran label=maxin max(v(in),10n,260n)

.tcl_wave tran FREQCKIN = FREQM(v(in), meas(maxin), meas(minin))

.defwave tran CKOUTREF='W(FREQCKIN)/8'
.plot tran V(in) W(FREQCKIN) W(CKOUTREF)
```

.TEMP

Set Circuit Temperature

```
.TEMP TS {TS}
```

The **.TEMP** command may be used to execute several successive simulations at various temperatures. The **.TEMP** command works for all analysis types. All input data for Eldo is assumed to have been measured at 27°C. Eldo also assumes a nominal temperature of 27°C unless a **TNOM** statement is present in the **.OPTION** command.

Note

Any model temperature defined by the **TMOD** parameter has priority over the **.TEMP** command, and furthermore, the **T** parameter has priority over both **TMOD** and **.TEMP**, that is, **T** has the highest priority.

.MPRUN can be used to take advantage of multi-processor machines for the **.TEMP** command. **.ALTER** has a higher priority than **.TEMP** when used with **.MPRUN**. Please see ".MPRUN" on page 729 for further information.

Parameters

- TS

  The temperature(s) for circuit simulation.

Example

```
.temp 0 27 60
```

Specifies circuit analyses at 0, 27 and 60 °C.

.TF

Transfer Function

```
.TF OV IN
```

The `.TF` command causes the small signal Transfer Function to be calculated by linearizing around a bias point. The gain from `IN` to `OV` is output along with the input and output impedances.

Parameters

- `IN`

  Input voltage source name. Must be an independent source.

- `OV`

  Requests the output voltage of a specific node or current through a voltage source. The syntax is as follows:

  V(N1[, N2]**)**
  Specifies the voltage difference between nodes `N1` and `N2`. If `N2` and the preceding comma are omitted, ground is assumed.

  I(Vxx[, **V**yy]**)**
  Specifies the current difference between the voltage sources **V**xx and **V**yy. If **V**yy and the comma are omitted, the current through **V**xx is output.

Example

```
* voltage source definition
vin 1 0 5
...
.tf v(3) vin
```

Specifies that the small signal Transfer Function be calculated for the voltage at node `3` with respect to the independent voltage source `vin`.

.TITLE

Set Title of Binary Output File

```
.TITLE name
```

The first line of the *.cir* file provides the title of the binary output file (*.wdb*). However, under a graphical environment the user is not supposed to manually modify the netlist, so this command can be used to define the title of the binary output file.

.TITLE also affects the different banners that appear in the ASCII output file.

.TOPCELL

Select the TOP Cell Subcircuit

```
.TOPCELL [=] <SUBCKT_NAME>
```

Usually, an automatic extraction tool is used to generate a hierarchical Spice netlist. However, in order to simulate the design, it may be necessary to add an X instance of the TOPCELL subcircuit, and re-specify all interface nodes. In these cases it is easier to use **.TOPCELL** SUBCKT_NAME, when the associated **.SUBCKT SUBCKT_NAME** and the **.ENDS** command will be ignored by the Eldo parser.

Note

The command **.TOPCELL** must be specified before any **.SUBCKT** command.

.TRAN

Transient Analysis

Point-Driven Analysis

```
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP DATA=dataname] [UIC] [MONTE=val]
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP parameter_name
+ TYPE nb start stop] [UIC] [MONTE=val]
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP parameter_name start stop incr]
+ [UIC] [MONTE=val]
```

Parameterized Analysis

```
.TRAN INCRn Tn [{INCRn Tn}] [TSTART=val] [SWEEP DATA=dataname]
+ [UIC] [MONTE=val]
.TRAN INCRn Tn [{INCRn Tn}] [TSTART=val] [SWEEP parameter_name
+ TYPE nb start stop] [UIC]  [MONTE=val]
.TRAN INCRn Tn [{INCRn Tn}] [TSTART=val] [SWEEP parameter_name
+ start stop incr] [UIC] [MONTE=val]
```

Data-Driven Analysis

```
.TRAN DATA=dataname [SWEEP DATA=dataname] [UIC] [MONTE=val]
.TRAN DATA=dataname [SWEEP parameter_name TYPE nb start stop]
+ [UIC] [MONTE=val]
.TRAN DATA=dataname [SWEEP parameter_name start stop incr]
+ [UIC] [MONTE=val]
```

This command activates a transient analysis. Transient output variables (that is, those variables contained within **.PRINT** and **.PLOT** commands in the input description file) are calculated as a function of time over a user specified time interval. The initial conditions are automatically determined by a DC analysis (unless the **UIC** parameter is specified) with all sources that are not time dependent being set to their DC values.

The Integral Equation Method (IEM) is used in **.TRAN** to solve FNS functions. The algorithm is just used for FNS, and not for the rest of the circuit. **.OPTION NOFNSIEM** reverts to the previous implementation based on state variables.

Multi-threading can be activated for a single DC or TRAN simulation, Eldo will share computer resources on a multi-processor machine. Alternatives are:

- command line flag **-mthread** (see “-mthread” on page 50) at Eldo invocation, or option **MTHREAD** in the netlist. Eldo will make use of all the possible CPUs on the machine.
- command line flag **-usethread #** (see “-usethread val” on page 56) at Eldo invocation, or option **USETHREAD**=val. This forces Eldo to use at maximum the specified (#) number of CPU. The number specified can exceed the number of CPUs available, but this is not recommended, even though Unix will allow it.

Statistics, generated at the end of simulation, show how many CPUs have been used for the current simulation. This number will also be printed out at the beginning of the TRAN simulation.

Parameters

- `TPRINT`

  The time interval used for the printing or plotting in the ASCII output *.chi* file of the results of transient analysis (in seconds). Also used to compute a default `HMAX` value in case the circuit does not contain any signals (no `PWL/SIN`, and so on), which is often the case in oscillator circuits. `TPRINT` can be specified as a parameter or as an expression.

- `TSTOP`

  The transient analysis duration in seconds. This can be specified as a parameter or as an expression.

- `TSTART`

  Start time for printing or plotting. No outputs are stored from 0 to `TSTART` seconds. This can be specified as a parameter or as an expression.

- `HMAX`

  Sets the maximal internal timestep. When `HMAX` is specified both in the **`.OPTION`** command and in the **`.TRAN`** command, the `HMAX` in **`.OPTION`** is considered by Eldo. See **`.OPTION`** "HMAX=VAL" on page 965.

- **`DATA`**`=dataname`

  Used in conjunction with the **`.DATA`** command. The `dataname` parameter should be specified using the **`.DATA`** command. Please refer to ".DATA" on page 585 for more information.

- **`UIC`**

  Use initial conditions. Eldo does not solve the quiescent operating point before beginning the transient analysis. Eldo automatically initializes all the node voltages itself as well as any user defined initial node voltages included in a **`.IC`** command. The **`UIC`** option is recommended for the simulation of astable or very large digital circuits.

- **`MONTE`**`=val`

  Monte Carlo analysis. Equivalent to **`.MC`** `val`. The syntax allows a different **`MONTE`** value for each run. However, the actual implementation in Eldo does not account for that: it is the last **`MONTE`** value to be specified in the netlist which will be taken into account.

-compat flag

When Eldo is run with the `-compat` flag, if the **`.TRAN`** command has four parameters, for example:

```
.TRAN tprint tstop tstart hmax
```

- if value4 (`hmax`) < value2 (`tstop`), it is treated as in Eldo standard mode:

  ```
  .TRAN tprint tstop tstart hmax
  ```

- if value4 (`hmax`) ≥ value2 (`tstop`), it is treated as a list of `INCRn Tn` values:

  ```
  .TRAN INCR1 T1 [{INCRn Tn}] [TSTART=val] [UIC]
  ```

- `INCR1`, ...`n`

  Used in the **`.PRINT`**/**`.PLOT`** command for printout purposes only. Between 0 and `T1`, a value will be printed for each `INCR`, and so on.

- `Tn`

  Simulation end time.

Sweep Parameters

This section contains **`SWEEP`** related parameters that are previously unspecified in the Parameters section.

- **`SWEEP`**

  Specifies that a sweep should be performed on a parameter or device name.

Note

`.TRAN` ... **`SWEEP`** will not work if **`.STEP PARAM`** is specified and a warning message will be issued stating that these two ways for specifying a sweep are not compatible.

- `parameter_name`

  Name of the parameter or device name to be swept.

- **`TYPE`**

  Can be one of the following:

  `DEC`
  Keyword to select logarithmic variation.

  `OCT`
  Keyword to select octave variation.

  `LIN`
  Keyword to select linear variation.

POI
Keyword to select a list of frequency points. **POI** is the same as **LIST** except that **POI** expects the number of points nb to be specified as it's first argument.

INCR
Increment of the parameter or device name to sweep. When **INCR** is specified as the TYPE parameter, the value which directly follows (nb) is the incrementing value.

- nb

  Number of points required, for example:

  ```
  .TRAN 1n 10n SWEEP P1 POI 3 1k 10k 100k
  ```

- start

  Start value of the parameter or device.

- stop

  Stop value of the parameter or device.

- incr

  Increment of the parameter or device name to sweep.

Note

When **INCR** is specified as the TYPE parameter, the value which directly follows (nb) is the incrementing value. If **INCR** is not specified, the incrementing value (incr) must be placed after the start and stop values.

For a circuit not containing any signals (no PWL/SIN, and so on), which is often the case in oscillator circuits, the value of TPRINT is also used to compute a default HMAX value. In such circuits, unless a **.OPTION HMAX** command was supplied, or a very low EPS value was given, Eldo would sometimes not detect the transitions that trigger oscillations, or the transitions that would cause them to die out, and designers who are familiar with Spice-like simulators would not understand the reason for Eldo not behaving in a Spice-like way. The reason for such a difference is only because Spice-like simulators use TPRINT as a HMAX value, while in Eldo, this was not the case.

The scenario inside Eldo is now the following: if the circuit does not contain any stimuli, then the HMAX value is either TPRINT, or TSTOP/DENOM, whichever value is larger. DENOM is based on EPS; DENOM is 50 by default, and 100 for EPS < 100U.

Examples

```
.tran .2u 40u sweep r1 INCR 5 100 5k
.tran .2u 40u sweep r 100 5k 5
```

The two lines above are equivalent. Either can be used to specify a transient analysis from 0 to 40us with print time intervals of 0.2us. The sweep specification forces Eldo to carry out a transient analysis on each value of r1 starting at 100Ω and stopping at 5kΩ with an incrementing value of 5Ω.

```
.tran 1ns 100ns
.plot tran v(2)
```

Specifies a transient analysis from 0 to 100 ns with a print step of 1 ns. The results of the transient analysis are plotted for the voltage at node 2 of the circuit within the limits specified in the **.TRAN** command.

```
.tran 2ns 100ns 50ns uic
.plot tran v(4)
```

Specifies a transient analysis from 0 to 100 ns and a print-out from 50 to 100 ns with a print step of 2 ns. The **UIC** option is also specified indicating that a DC analysis will not be performed and that initial voltage will be set up automatically. The results of the transient analysis are plotted for the voltage at node 4 of the circuit within the limits specified in the **.TRAN** command.

Examples of this type of analysis can be found in "Tutorials" on page 1341.

Parameters are allowed in transient analysis commands as shown in the following example:

```
.param p1 = 1e9
.tran dec 10 1 p1
```

NOISE analysis may also be run from within a **.TRAN** command, see "AC in the middle of a .TRAN" on page 543 for more details.

.TSAVE

Save Simulation Run at Multiple Time Points

```
.TSAVE [REPLACE|NOREPLACE] TIME=VALUE [FILE="fileBasename"]
```

The **.TSAVE** command will save the state of the simulation at a specified time point. The state of the simulation is saved to a *.iic* file. The file can be used to restart the simulation from the specified time point using the **.RESTART** command. See ".RESTART" on page 854. The state of the simulation can be saved at more than one time point by specifying the **.TSAVE** command for each time point.

For saving a simulation run without specific multiple time points, see ".SAVE" on page 857.

Parameters

- **REPLACE**

  All previously saved checkpoint files in the output directory will be removed and replaced with the checkpoint file specified. Default. Optional.

- **NOREPLACE**

  Only the checkpoint file with the same name will be modified, all the remaining checkpoint files will remain unchanged. The checkpoint file will be saved in the output directory. Optional.

- **TIME**=VALUE

  Specifies the time at which the simulation will be saved. Mandatory.

- **FILE**="fileBasename"

  Specifies the first part of the checkpoint file name. The file name will take the form *fileBasename_timepoint.iic*, where *timepoint* is the time that the simulation was saved. The file will contain the information from the simulation run for the specified time point. If omitted Eldo will save the file with the name of the top-netlist: if the top-netlist is called *spice-on-top.cir* Eldo will use the name *spice-on-top_...*. Optional.

Examples

```
.TSAVE TIME=200ns FILE="spice_ontop"
```

The state of the simulation will be saved at 200ns. All previously saved checkpoint files in the output directory will be removed and replaced with the checkpoint file *spice_ontop_2.000000E-7.iic*.

```
.TSAVE NOREPLACE TIME=200ns FILE="spice_ontop"
```

The state of the simulation will be saved at 200ns. All previously saved checkpoint files in the output directory will remain unchanged if the filename is unique. The checkpoint file will be saved with the name *spice_ontop_2.000000E-7.iic*.

```
.PARAM sim=250ns
.PARAM chkpnt_file='"timepoint"'
```

```
.TSAVE noreplace time=sim file=$(chkpnt_file)
```

The state of the simulation will be saved at 250ns as defined on the parameter `sim`. The checkpoint file name will be *timepoint* as defined by the string parameter `chkpnt_file`.

```
.TSAVE NOREPLACE TIME=10ns FILE="spice_ontop"
.TSAVE NOREPLACE TIME=100ns FILE="spice_ontop"
.TSAVE NOREPLACE TIME=1000ns FILE="spice_ontop"
```

The state of the simulation will be saved at 10ns, 100ns and 1000ns to three independent files. If `noreplace` was not specified on the last **`.TSAVE`** command the checkpoint files saved at 10ns and 100ns would be removed.

.TVINCLUDE

Test Vector Files

.TVINCLUDE [**FILE**=]FILENAME [**COMP**=ON|OFF] [**ERRNODE**[=YES|NO]]

Test vectors can be included in a netlist. This file allows the user to define a bus, specify inputs and check output values. It is possible to compare simulation results using test vectors. A test vector is an external file containing a record of circuit stimulus and response.

Parameters

- FILE=

  Specifies the filename. Optional.

- FILENAME

  Test Vector filename.

- COMP=ON|OFF

  Define whether the output vector is compared to simulation results. Default is ON. Optional.

- ERRNODE[=YES|NO]

  If set to YES (default value), an error is printed for signals using undeclared nodes. If set to NO, the following warning is displayed:

  ```
  Warning 445: COMMAND .TVINCLUDE: node %s not found (%s).
  Test vector specifications ignored for this node.
  ```

Boundaries are set by using the following options:

.OPTION LOWVOLTAGE=VAL **HIGHVOLTAGE**=VAL **LOWVTH**=VTH1 **HIGHVTH**=VTH2

- **LOWVOLTAGE** and **HIGHVOLTAGE** are used for input signals
- **LOWVTH** and **HIGHVTH** are used for output checking
- Default values are: **LOWVOLTAGE**=0, **HIGHVOLTAGE**=5, **LOWVTH**=2.4, **HIGHVTH**=2.6

Note

For more details regarding the setting of boundaries with these options, please refer to "HIGHVOLTAGE=VAL" on page 977.

Test Vector File Format

Figure 10-6 shows the contents of a test vector file for a two-input AND gate.

Figure 10-6. Example Test Vector File

```
/*
 *   test vector file for a
 *   two-input AND gate sampled
 *   at various intervals.
 */
CODEFILE
UNITS pS
RISE_TIME 50
FALL_TIME 50
INPUTS  in[0], in[1], in[2], in[3];
OUTPUTS out(to=max);
CODING(ROM)
# start vectors
RADIX  <4>1;
@0     <0>1;
@1900  <a>0;
@5000  <2>0;
@6900  <e>0;
@10000 <6>1;
@11900 <7>0;
@15000 <3>0;
@16900 <b>0;
@20000 <1>1;
END
```

A test vector file consists of the following parts:

- Header
  The header specifies the units of time used in the test vectors and the direction and order of the inputs and outputs. Refer to “Header” on page 920.
- Comments in Test Vector Files
  Comments can appear anywhere in the file. Refer to “Comments in Test Vector Files” on page 921.
- Radix
  Test vectors can have a radix value of 1, 2, 3 or 4. By default the value is 1.
- Test Vectors
  Test vectors consist of a time stamp followed by the input and output signal data at the time indicated by the time stamp. Refer to “Test Vectors” on page 921.

Note

There is no provision for line continuation in test vector files. Test vector file line lengths are not limited.

Header

The keyword **CODEFILE** marks the beginning of the header; the keywords **CODING(ROM)** mark the end of the header. Within the header are three classes of statements:

- **UNITS** Statement

  Specifies the units used for time stamps (the time units of measure) in the test vectors. Format the **UNITS** statement as follows:

  ```
  UNITS units
  ```

  `units`

  Must be fs (femtoseconds), ps (picoseconds), ns (nanoseconds), us (microseconds), or ms (milliseconds). This optional parameter defaults to fs. Units are case insensitive.

- **RISE_TIME** and **FALL_TIME** Statements

  Specifies the rise and fall times for digital signals. The optional **RISE_TIME** and **FALL_TIME** statements are single-parameter declarations of the digital signal transition durations. If you omit the **RISE_TIME** and **FALL_TIME** statements, Eldo uses the DefaultRiseTime and DefaultFallTime simulation option values (default 1e-10). Format the statements as follows:

  ```
  RISE_TIME n
  FALL_TIME n
  ```

  `n`

  Specifies a digital transition rise or fall time. The **UNITS** statement specifies the units of time (fs, ps, ns, or us).

  The **RISE_TIME** and **FALL_TIME** definitions affect the timing of signals moving from one state to another. Refer to “Timing of Changing States” on page 922.

- **INPUTS** and **OUTPUTS** Statements

  Defines the direction and order of the signals in the test vectors. The **INPUTS** and **OUTPUTS** statements can be declared on single or multiple lines. Signals in the input and output lists can have one parameter denoting a time offset. A signal specification has the following general form:

  ```
  INPUTS signal [(to=n])]...;
  OUTPUTS signal [(to=n])]...;
  ```

  `signal`

  A Spice netlist signal name. Specify bidirectional signals in both **INPUTS** and **OUTPUTS** statements.

`to=n`

Specifies a time offset (to) that applies to both inputs and outputs. This parameter causes input forcing or output comparison to be delayed by ***n*** time units (fs, ps, ns, µs, or ms). The UNITS Statement specifies the units of time.

The time offset is relative to the time the vector is applied. For output signals, if ***n*** is replaced by the keyword **max**, then the comparison occurs one femtosecond before Eldo reads the next vector.

If no time offsets are specified, inputs start being forced at the vector time.

`; (semicolon)`

Terminates the **INPUTS** or **OUTPUTS**.

In the test vector file example illustrated in Figure 10-6 on page 919, all the inputs are forced at the same time, with no offsets. Output comparisons are delayed until just before the next vector is forced.

Comments in Test Vector Files

Comments can appear anywhere in a test vector file. They can take the form of C-style comments (delimited by **/*** and ***/**) or shell-style comments (prefixed by a **#** character). Comments that appear within the header are ignored when the test vector file is read.

Radix

Test vectors can have a radix value of 1, 2, 3 or 4. By default the value is 1. Table 10-32 lists the different radix values and their meanings:

Table 10-32. Test Vector Radix Values

Value	Radix
1	Binary
2	Decimal
3	Octal
4	Hexadecimal

Test Vectors

The vector portion of the test vector file starts after the header and ends with the keyword **END**. Format test vectors according to the following form:

```
@timestamp <input_data >output_data ;
```

`timestamp`

Specifies the simulation time in at which the vector was generated. Supply an integer value (with no decimal). The UNITS Statement specifies the units of time. Eldo applies the vector at this time during verification.

`input_data`

Specifies the states of the inputs at the simulation time shown by the timestamp. The less than symbol ‘<’ precedes the input portion of the vector. Refer to Table 10-33 on page 922 for valid state characters.

`output_data`

Specifies the states of the outputs at the simulation time shown by the timestamp. The greater than symbol ‘>’ precedes the output portion of the vector. Refer to Table 10-33 on page 922 for valid state characters.

`; (semicolon)`

Terminates the vector.

The input and output portions of the vector contain columns of state characters. Each column is associated with one signal, in the order defined in the input and output lists in the header. If a signal is bidirectional, it could have one column in the input portion of the vector and one column in the output portion of the vector.

Table 10-33 lists the different state characters and their meanings:

Table 10-33. Test Vector State Characters

State *	Meaning for Inputs	Meaning for Outputs
0 or **L**	Force LOW	Verify LOW
1 or **H**	Force HIGH	Verify HIGH
X	Ignore; do not force	Do not verify
. or **Z**	Do not force	Do not verify

* Logic level states (L, H, X, and Z) are case insensitive in Eldo.

Timing of Changing States

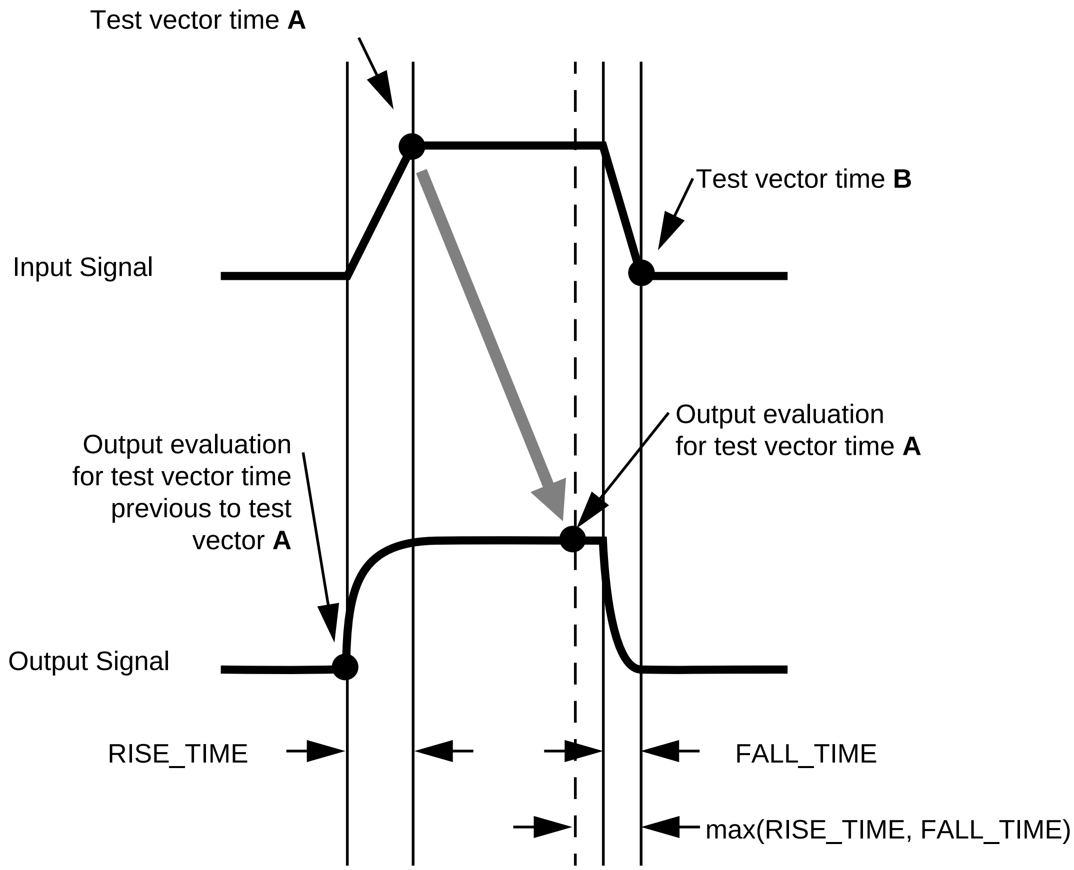
Eldo imposes the following timing rules while changing signal states specified by test vectors:

- **Forced state to forced state**
  A signal changing from a forced state (non-Z) to another forced state begins its transition **RISE_TIME** (or **FALL_TIME**) before the vector time so that the signal reaches the new value at the vector time.

- **High impedance to forced state**
  A signal changing from high impedance state (Z) to a forced state starts to be forced **RISE_TIME** (or **FALL_TIME**) before the vector time.

- **Forced state to high impedance**
  A signal changing from a forced state to a high impedance state (Z) is tri-stated (released) at the vector time.
- Eldo uses the greater value of the **RISE_TIME** and **FALL_TIME** definitions in determining output evaluation timing (refer to Figure 10-7). The input signals always arrive at the evaluation time. However, if the **RISE_TIME** and **FALL_TIME** values are different, one evaluation will be performed some time before the transition begins.

Figure 10-7. Test Vector Output Evaluation



Note

Because Eldo determines the test vector output evaluation based on the maximum value of **RISE_TIME** and **FALL_TIME**, caution should be exercised when assigning them different values.

The following output evaluations occur:

- The minimum time step (1fs) before the next transition begins if the next transition is determined by the *greater* value of **RISE_TIME** and **FALL_TIME**.

- Before the next transition begins by the difference between **RISE_TIME** and **FALL_TIME** if the next transition is determined by the *lesser* value of **RISE_TIME** and **FALL_TIME**.

.UNPROTECT

Netlist Protection

```
.UNPROTECT
```

This command is used in conjunction with ".PROTECT" on page 850 to exclude a section of a netlist from being copied into the output file. Can be specified to control the end of the encryption process with the Eldo Encryption tool.

Example

```
.PROTECT
vin 2 0 ac 0.5
r1 2 3 5k
c3 3 0 0.1p
.ac dec 10 10e+4 10e+8
.plot ac vdb(3)
.UNPROTECT
```

The lines in a netlist between the two commands **`.PROTECT`** and **`.UNPROTECT`** are not printed to the output file.

.USE

Use Previously Simulated Results

```
.USE FILE_NAME [NODESET|IC|GUESS|OVERWRITE_INPUT]
```

This command takes a set of voltages previously saved using the **.SAVE** <fname> DC command, and inserts these as **.NODESET**, **.GUESS** or **.IC** values at the point of the circuit where this command is present. It is also possible to have multiple occurrences of the **.USE** command in an input netlist. See ".SAVE" on page 857.

For more details, refer to the section ".SAVE, .USE & .RESTART" on page 1080 in the Speed and Accuracy chapter.

This command cannot be used with previously saved simulation results from a **.TSAVE** command.

Note

Care must be taken when using a *.sav* file generated in a previous release as this can cause incorrect simulation results. It is recommended that only *.sav* files generated with the current release are used.

Parameters

- FILE_NAME

  Filename into which the DC values were saved via the **.SAVE** <fname> DC command.

- **NODESET**

  Indicates that the read in voltages should be used as **.NODESET** values.

- **IC**

  Indicates that the read in voltages should be used as **.IC** values.

- **GUESS**

  Indicates that the read in voltages should be used as **.GUESS** values.

- **OVERWRITE_INPUT**

  Has the same effect as **GUESS**, however Eldo is allowed to change the DC values of input if they don't match the .ic file. This parameter allows the user to perform AC analysis close to a previous transient saved point (saving with the **.SAVE end** or **.SAVE time** command).

Note

If **.USE OVERWRITE_INPUT** is used in conjunction with **.AC UIC**, then the file referred to in the **.USE** statement will be used for each AC analysis, and not only for the first one as was the case in versions prior to v6.3. For example, if there is a **.STEP** command, then the *use file* will be used for each AC analysis triggered by the **.STEP**, and not only by the first one.

Examples

```
.use test1.exa nodeset
```

Specifies that DC values found in the file *<test1>.exa* should be read and used as **.NODESET** values.

```
.use circuit1.iic ic
```

Specifies that DC values found in the file *<circuit1>.iic* should be read and used as **.IC** values.

.USEKEY

Use Reliability Model Key (Password)

```
.USEKEY [MODEL=model_name] KEY=key_value
```

This reliability analysis command provides the encryption key (password) to be used to allow the UDRM API to retrieve encrypted model parameter's value.

For the complete description of this command and information on all reliability commands, see the separate chapter Reliability Simulation.

.USE_TCL

Use Tcl File

```
.USE_TCL FILENAME [MODE=PPL|EZWAVE]
```

This command loads the Tcl file `FILENAME` into the Eldo Tcl interpreter.

This allows you to dynamically manage User Defined Functions (UDF) written in Tcl language. The functions of the post-processing library (PPL), EZwave user-defined functions, and other commands are available through the Tcl interface.

Although the PPL and EZwave contain equivalent functions, the syntax to call the functions is different, and sometimes the function name and parameters are different. It is possible to write user-defined functions that can be used in both Eldo and EZwave.

After loading the Tcl file, Eldo can use all the functions written in this file as if they were macros. This means that these functions can accept any argument, are defined for the whole netlist, and can return both numbers and/or waves.

For performing a single call to a Tcl function, see the command “.CALL_TCL” on page 559. For further information on both commands, see the Post-Processing Library chapter of this manual.

Parameters

- `FILENAME`

  Filename of the Tcl file to be loaded.

- `MODE=PPL|EZWAVE`

  Select the mode for evaluating Tcl functions. `PPL` selects the post-processing library (PPL) functions. `EZWAVE` selects the EZwave user-defined functions. Default is `PPL`. Although the PPL and EZwave contain equivalent functions, the syntax to call the functions is different, and sometimes the function name and parameters are different.

Example

If *resi.tcl* contains:

```
proc PARAM_EVAL { a } {
   return [expr $a * 3]
}
```

and the netlist contains:

```
* comment
.use_tcl resi.tcl
v1 1 0 pwl(0 0 10n 10)
r1 1 0 r=PARAM_EVAL(2)
.tran 1n 10n
```

```
.plot tran i(r1)
.end
```

This example demonstrates how to use a very simple Tcl function in a SPICE-like netlist. **.USE_TCL** loads the file into an internal Tcl interpreter, making all functions known to Eldo. Thus resistor `r1` gets its value from the Tcl function `PARAM_EVAL` just as if `PARAM_EVAL` is a macro (defined with **.DEFMAC**).

.USE_VERILOGA

Load Compiled Verilog-A Modules

```
.USE_VERILOGA filename
```

Loads compiled Verilog-A modules in Eldo.

The **.USE_VERILOGA** command finds the specified Verilog-A compiled file and treats its behavior as a part of the Eldo netlist. If the specified Verilog-A compiled file does not exist then Eldo generates an error message.

Refer to the *Eldo Verilog-A User's Manual* for further information.

Parameters

- `filename`

  Name of the Verilog-A source file to be used by Eldo in a simulation.

.VEC

Test Vector Files

```
.VEC 'file_name'
```

Description

Loads the vector file, file_name, of HSPICE format. For this format of file, see Digital I/O Files in the *ADiT User's and Reference Manual*.

See also .TVINCLUDE.

.VERILOG

Compile and Load Verilog-A Modules

```
.VERILOG filename
```

Compiles and loads Verilog-A modules in Eldo.

To incorporate Verilog-A modules (contained in a Verilog-A source file) in an Eldo netlist, specify the source file name in a **.VERILOG** statement in the netlist.

By default the Verilog-A source file is compiled using the default Verilog-AMS compiler in Questa ADMS.

Refer to the *Eldo Verilog-A User's Manual* for further information.

Parameters

- `filename`

  Name of the Verilog-A source file to be compiled. `filename` may include a relative or absolute pathname.

.WCASE

Worst Case Analysis

```
.WCASE DC|AC|TRAN [OUTPUT=MIN|MAX|BOTH] [VARY=LOT|DEV|BOTH]
+ [TOL=VAL] [ALL] [SORT_REL=VAL] [SORT_ABS=VAL] [SORT_NBMAX=VAL]
```

Worst Case Analysis computes worst case values for waveform data extracted using the **.EXTRACT** command according to a set of parameters that have **LOT** and **DEV** variations.

For DC sensitivity analysis, the **.SENS** command can be used, see “.SENS” on page 863. For AC sensitivity analysis, the **.SENSAC** command can be used, see “.SENSAC” on page 865.

Worst Case Analysis uses the same **LOT** and/or **DEV** parameters as MC analysis to set these parameters to be varied.

Note

Multiple sensitivity and statistical analyses cannot be used simultaneously. Specifically, **.DCMISMATCH**, **.MC**, and **.WCASE** are exclusive. Only one of them can be specified in a netlist.

The keyword, **STATISTICAL**= 0|1, can be specified on X instances, device declarations, or on **.SUBCKT** definitions, to specify whether any statistical variation due to worst case analysis can be applied to the specified entities. If **STATISTICAL** is 0, the selected devices will keep their nominal values. If **STATISTICAL** is 1, the selected devices have statistical variation applied. The global default can be specified via option **STATISTICAL**= 0|1. Default is 1.

Different entities are able to share the same distribution. Anywhere Eldo accepts **LOT**/**DEV** specifications, you can specify **LOTGROUP**=group_name. Please refer to “.LOTGROUP” on page 701.

Considering a design which contains P parameters that have a statistical variation, and T targets (**.EXTRACT** commands), Eldo will perform several simulations:

1. The first simulation is the nominal simulation, using nominal values for the P parameters.
2. Next P simulations are sensitivity analysis runs, where Eldo applies a small variation to one of the P parameters, keeping all other parameters to their nominal value, and computes the sensitivity of the T targets relative to the current parameter.

   For each of the T targets, Eldo outputs a message like:

   ```
   With respect to MODEL R LOT
   Variation of MAX(V(1)) is negative: -1.000000e-02 (Unit/%) or
   1.000000e+00 (%/%)
   ```

Negative variation means that when parameter is increased, target is decreasing.

3. Then with all the sensitivity data, Eldo computes the combination of min or max parameter values that will generate the best and worst cases (**MIN** and **MAX**) for each of the `T` target.

   There can be at most 2×T worst case simulations. During these simulations, Eldo does not output anything.

4. Finally, Eldo computes the worst case information for all the targets, and outputs this information.

   Thus, there can be at most 1 + P + 2×T simulations. Worst Case analysis assumes the influence of each parameter to be linear, and not correlated with other parameters.

The `SORT_` parameters are used to limit the amount of output information.

Parameters

- **DC**

  Specifies a DC analysis. An analysis type MUST be specified.

- **AC**

  Specifies an AC analysis. An analysis type MUST be specified.

- **TRAN**

  Specifies a transient analysis. An analysis type MUST be specified.

> **.WCASE** can also be used in Steady-State analysis (**.SST**). For more information see .SST (Steady-State Worst Case analysis) of the *Eldo RF User's Manual*.

- **OUTPUT**

  Specifies the type of Worst Case Analysis:

  MIN
  Only the minimum case is extracted.

  MAX
  Only the maximum case is extracted.

  BOTH
  Both minimum and maximum cases are extracted. Default=**BOTH**.

- **VARY**

  Specifies the kind of variation; **DEV**, **LOT** or **BOTH**. Default=**BOTH**.

- **TOL**

  Specifies that a variation or tolerance is to be applied to the sensitivity measurement.

- VAL

  The percentage tolerance value for the sensitivity: `<value>`. Default=2%. If in the *.chi* file Eldo states that sensitivity is zero, this may be due to the **TOL** value being set too small.

- **ALL**

  Causes all intermediate waves will be plotted/printed. This can result in large output ASCII/Binary files, depending on the number of waves and the number of Worst Case runs.

- SORT_REL

  Only devices with a contribution to the output greater than the value: SORT_REL×<MAX_variation_on_output> will be listed. The default value of SORT_REL is 0.001, which means that all devices contributing to more than 1/1000 of the maximum variation will be listed.

- SORT_ABS

  Used to specify the absolute threshold, below which contributors will not be listed. Default value is 0.

- SORT_NBMAX

  Allows the user to limit the list of contributors to a certain value. By default, all contributors are listed.

Note

The sorting parameters are additive, that is, their respective effects are cumulative. For example, to create a report with only devices contributing to 10% or more of the maximum output deviation and have 15 contributors at most you would specify:

SORT_REL=0.1
SORT_NBMAX=15.

Related options

CARLO_GAUSS (see "CARLO_GAUSS" on page 975), **SIGTAIL** (see "SIGTAIL=VAL" on page 982), **STATISTICAL** (see "STATISTICAL=0|1" on page 982), **DISPLAY_CARLO** ("DISPLAY_CARLO" on page 999)

Example

```
Worst Case analysis example
* circuit: RC filter
r1 in out rmod 10k
c1 out 0 cmod 1p
vin in 0 ac 1
.model rmod res dev=10%
.model cmod cap dev=10%
* extract cutoff frequency at –3 dB
.extract xycond (xaxis, vdb(out)<=max(vdb(out))–3)
.wcase ac
.ac dec 10 1 1G
.plot ac vdb(out) vp(out)
.end
```

The following results will be obtained in the *.chi* file:

Sensitivity:

```
With respect to MODEL RMOD DEV: Object :R1
   Variation of  XYCOND(XAXIS, ...) is negative: –1.598533e+05 (Unit/%)
With respect to MODEL CMOD DEV: Object :C1
   Variation of  XYCOND(XAXIS, ...) is negative: –1.598533e+05 (Unit/%)
```

Worst case values:

```
*XYCOND(XAXIS, ...) NOM: 1.587800E+0  MIN: 1.312462E+0 MAX: 1.959982E+07
```

Note

If incorrectly used, run times for this command may become excessive, especially where results are completely out of range. **`LOT/DEV`** parameters must be applied correctly to devices/models to get reasonable results.

.WIDTH

Set Printer Paper Width

```
.WIDTH OUT=80|132
```

Sets the paper width in number of characters of the output print device.

When multiple occurrences of the **.WIDTH** command are specified, a warning is issued. The last **.WIDTH** command will override all previous commands.

Parameters

- **OUT**=80

  Specifies that the number of columns or characters on the output device is 80.

- **OUT**=132

  Specifies that the number of columns or characters on the output device is 132. The default value is 132.

Example

```
.width out=80
```

Sets the output width on the output device to 80 columns.

```
.WIDTH  OUT=80
.WIDTH  OUT=132
```

Sets the output width on the output device to 132 columns, as the previous command is ignored.

Chapter 11
Simulator and Control Options

Introduction

The **`.OPTION`** command allows the user to modify Eldo execution behavior by allowing the setting of parameter values other than the default ones.

Multiple **`.OPTION`** commands can be used in a netlist. A single **`.OPTION`** statement can contain multiple options in any combination and any order.

If an option is stated more than once, the last specified value is used. Unless otherwise described, if an option is not stated, it defaults to zero.

Eldo Options

.OPTION

```
.OPT[ION] OPTION[=VAL] {OPTION[=VAL]}
```

The option descriptions have been divided into sections according to what area of the program they affect.

Note
Options from the SPICE language that are not included in the following list are ignored by Eldo.

Parameters

- `OPTION[=VAL]`

  The following tables show a list of the possible options which can be used with the **`.OPTION`** command. Click on an option to jump to the detailed description of that option. Each option is described in greater detail in this chapter.

  The first table lists all the options in alphabetical order. The following tables list the options divided into categories.

Table 11-1. Eldo Options

ABSTOL	ABSVAR	ACCSEMICOL	ACDERFUNC
ACM	ACOUT	ACSIMPROG	ADJSTEPTRAN
ADMS_FAST_PARSE	ADMSBS	AEX	AIDSTP
ALIGNEXT	ALTER_ADDSTEP	ALTER_NOMINAL_ TEXT	ALTER_SUFFIX
ALTERELDO	ALTINC	AMMETER	ANALOG
ASCII	ASCII=val	ASCIIPLOT	ASPEC
AUTOSTOP	AUTOSTOPMODULO	BE	BLK_SIZE
BLOCKS=IEM	BLOCKS=NEWTON	BSIM3VER	BSLASHCONT
CAPANW	CAPTAB	CARLO_GAUSS	CHECKDUPL
CHGTOL	CKDCPATH	COLLAPSE_DSPF_ OUTPUT	COMPAT
COMPEXUP	COMPMOD	COMPNET	CONTINUE_INCLUDE
CONTINUOUS_FFT	COU	CPTIME	CSDF
CSHUNT	CTEPREC	D2DMVL9BIT	DCLOG
DCPART	DCSIMPROG	DEFA2D	DEFAD

Table 11-1. Eldo Options

DEFAS	DEFAULTFALLTIME	DEFAULTRISETIME	DEFCONVMSG
DEFD2A	DEFL	DEFNRD	DEFNRS
DEFPD	DEFPS	DEFPTNOM	DEFRMSNTR
DEFW	DICPRIO	DIGITAL	DISPLAY_CARLO
DOTNODE	DPTRAN	DSCGLOB	DSPF_LEVEL
DUMP_EXTRACT	DUMP_FILE_LIST	DUMP_MCINFO	DVDT
DYND2ALOG	DYND2ALOG2	ELDOMOS	EMPTY_MCHISTO
ENGNOT	EPS	EPSO	ERR0DIV0
EXTCGS	EXTERR	EXTFILE	EXTMKSA
EXTMOD_GENWAVE	EXTRACT_EVAL_FINAL	EXTRACT_VECT_AXIS	FALL_TIME
FASTRLC	FLICKER_NOISE	FLOATGATE0	FLOATGATECHECK
FLOATGATERR	FLUXTOL	FNLEV	FREQSMP
FROM_TO	FS_PARTITIONING	FS_PARTITION_DEBUG	FS_SOLVE_AMS_NODES
FSDB	FSDB_SINGLE_FILE	FT	GEAR
GENK	GMIN	GMIN_BJT_SPICE	GMINDC
GNODE	GRAMP	GSHUNT	HACC
HIER_SCALE	HIGHVOLTAGE	HIGHVTH	HISTLIM
HISTO_ZERO	HMAX	HMIN	HRISEFALL
IBIS_SEARCH_PATH	ICDC	ICDEV	IEM
IKF2	INCLIB	INFODEV	INFOMC
INFOMOD	INGOLD	INPUT	INTERP
ITL1	ITL3	ITL4	ITL6
ITL7	ITL8	ITOL	ITRPRT
JTHNOISE	JWDB	JWDB_ACTRAN_USE_TIME	JWDB_EVENT
JWDB_EXTENSIONS	JWDB_PERCENT	KEEP_DSPF_NODE	KEEP_HMPFILE
KEEPDANGLING	KEEPSHORTED	KLIM	KWSCALE
LCAPOP	LIBINC	LICN	LIMNWRMOS
LIMPROBE	LIST	LOCAL_NOWARN	LOOPV0
LOWVOLTAGE	LOWVTH	LVLTIM	LVS_IGNORE_VARIABLE
M53	MACMOD	MAX_CHECKBUS	MAX_DSPF_PLOT

Table 11-1. Eldo Options

MAXADS	MAXL	MAXNODEORD	MAXNODES
MAXORD	MAXPDS	MAXSTEP	MAXTOTWARN
MAXTRAN	MAXV	MAXW	MAXWARN
MC_IGNORE_BINNING	MC_NOMINAL_OP	MEAS_TARGWHEN	MEASFILE
METHOD=GEAR	MINADS	MINL	MINPDS
MINRACC	MINRESISTANCE	MINRVAL	MINW
MIXEDSTEP	MMSMOOTH	MMSMOOTHEPS	MNUMER
MOD4PINS	MODMONTE	MODWL	MODWLDOT
MOTOROLA	MSGBIAS	MSGNODE	MTHREAD
NETSIZE	NEWACCT	NEWTON	NGATEDEF
NGTOL	NMAXSIZE	NO_FS_VA	NOACDERFUNC
NOACT0	NOADMSBS	NOAEX	NOALTINCEX
NOASCII	NOASCIIPLOT	NOAUTOCTYPE	NOBOUND_PHASE
NOBSLASHCONT	NOCATMX	NOCKRSTSAVE	NOCMPUNIX
NOCONVASSIST	NOCOU	NODCINFOTAB	NODCPART
NODCPOWNEG	NODE	NODEFNEWTON	NODEFRMSNTR
NODUPINSTERR	NOELDOLOGIC	NOELDOSWITCH	NOERR_ XPINSMISMATCH
NOEXTRACTCOMPLEX	NOFNSIEM	NOICNODE	NOIICXNAME
NOINIT	NOISE_SGNCONV	NOJWDB	NOKEYWPARAMSST
NOKWSCALE	NOLAT	NOLICN	NOLTEDISC
NOMATSING	NOMEMSTP	NOMOD	NOMODWL
NONOISE	NONWRMOS	NOOP	NOPAGE
NOPROBEOP	NOQTRUNC	NORMOS	NOSETBUSEXPAND
NOSIZECHK	NOSMKMCWC	NOSSTKEYWORD	NOSTATP
NOSWITCH	NOTRC	NOTRCLIB	NOVATOPOCHK
NOWARN	NOWAVECOMPLEX	NOXTABNOISE	NOZSINXX
NSAFILE_FORMAT	NUMDGT	NWRMOS	OPALLDC
OPSELDO_ABSTRACT	OPSELDO_DETAIL	OPSELDO_DISPLAY_ GOALFITTING	OPSELDO_FORCE_ GOALFITTING
OPSELDO_JWDB_RUN	OPSELDO_NETLIST	OPSELDO_ NO_DUPLICATE	OPSELDO_ NOGOALFITTING
OPSELDO_OUTER	OPSELDO_OUTPUT	OPTYP	OSR
OUT_ABSTOL	OUT_REDUCE	OUT_RELTOL	OUT_RESOL

Table 11-1. Eldo Options

OUT_SMP	OUT_STEP	PARAM_BEFORE_USE	PARAMETRIC_ ACTRAN
PARAMOPT_ NOINITIAL	PARHIER	PARTGATE_AMS_ALL	PARTVDD
PARTVDD_AMS_ALL	PATTERN_MAX_ ALLOWED_COEFF	PCS	PCSPERIOD
PCSSIZE	PEVFLY	PGATEDEF	PIVCHECK
PIVREL	PIVTOL	PODEV	POST
POST_DOUBLE	POWNEG0	PRINT_ACOP	PRINTFILE_STEP
PRINTFILE_FREQ_ STEP	PRINTFILE_TIME_ STEP	PRINTLG	PROBE
PROBEOP	PROBEOP2	PROBEOPX	PSF
PSF_ALL_FILES	PSF_FULLNAME	PSF_NODEVICE_ NOISE	PSF_SCALARDC
PSF_VERSION	PSF_WRITE_ALL	PSFASCII	PSOSC
PSTRAN	QTRUNC	QUOTREL	QUOTSTR
RAILINDUCTANCE	RAILRESISTANCE	RANDMC	RATPRINT
REDUCE_KEEP_INST	REDUCE_KEEP_NODE	REDUCE_KEEP_ OUTPUTS	REDUCE_MAX_CAP
REDUCE_MAX_IND	REDUCE_MAX_RES	REDUCE	RELTOL
RELTRUNC	RELVAR	RESET_MULTIPLE_R UN	RESNW
RGND	RGNDI	RISE_TIME	RMMINRVAL
RMOS	RSMALL	RZ	SAMPLE
SAVETIME	SCALE	SCALEBSIM	SCALM
SDA	SEARCH	SHRINK_FACTOR	SIGTAIL
SIMUDIV	SLASHCONT	SMOOTH	SOIBACK
SPI3ASC	SPI3BIN	SPI3NOCOMPLEX	SPICEDC
SPIOUT	SPLITC	SPMODLEV	STARTSMP
STAT	STATISTICAL	STEP	STOPONFIRSTERROR
STRICT	SUBALEV	SUBFLAGPAR	TEMP_UNIT
TEMPCOUK	THERMAL_NOISE	TIMEDIV	TIMESMP
TMAX	TMIN	TNOM	TPIEEE
TRAP	TRTOL	TUNING	ULOGIC
UNBOUND	USEDEFAP	USE_LOCATION_MAP	USE_SPECTRE_ CONSTANT

Table 11-1. Eldo Options

USEFIRSTDEF	USETHREAD	VBCSAT	VERBOSE
VMAX	VAMAXEXP	VMIN	VNTOL
VOLTAGE_LOOP_ SEVERITY	VXPROBE	WARN2ERR	WARNING_ DEVPARAM
WARN	WARNMAXV	WBULK	WDB_IDELTA
WDB_VDELTA	WDB_NOSYNCHRO	WL	WRITE_ALTER_ NETLIST
WSF	WSFASCII	XA	XBYNAME
YMFACT	ZCHAR	ZDETECT	ZOOMTIME

Options divided into categories:

Table 11-2. Simulator Compatibility Options

COMPAT	COMPMOD	COMPNET	MOTOROLA
SDA	SPI3ASC	SPI3BIN	SPI3NOCOMPLEX
SPICEDC	SPIOUT	USE_SPECTRE_ CONSTANT	WSF
WSFASCII			

Table 11-3. Netlist Parser Control Options

ACCSEMICOL	ADMS_FAST_PARSE	ADMSBS	ALTER_ADDSTEP
ALTERELDO	ALTINC	BSLASHCONT	CHECKDUPL
CKDCPATH	COMPEXUP	CONTINUE_INCLUDE	DICPRIO
DOTNODE	ERR0DIV0	EXTERR	KEEPDANGLING
KEEPSHORTED	LOOPV0	MACMOD	MEAS_TARGWHEN
MTHREAD	NOALTINCEX	NOACT0	NOADMSBS
NOBSLASHCONT	NOCMPUNIX	NOELDOLOGIC	NOERR_ XPINSMISMATCH
NOKEYWPARAMSST	NOMATSING	NOSETBUSEXPAND	NOSSTKEYWORD
NOZSINXX	PARAM_BEFORE_USE	PARHIER	PATTERN_MAX_ ALLOWED_COEFF
PEVFLY	POWNEG0	QUOTREL	QUOTSTR
RMV0	SLASHCONT	STOPONFIRSTERROR	STRICT
SUBALEV	SUBFLAGPAR	USEFIRSTDEF	USETHREAD
USE_LOCATION_MAP	VOLTAGE_LOOP_ SEVERITY	WARN2ERR	XBYNAME

Table 11-4. Simulation Speed, Accuracy and Efficiency Options

ABSTOL	ABSVAR	ADJSTEPTRAN	AIDSTP
CAPANW	CHGTOL	DVDT	EPS
FASTRLC	FLUXTOL	FREQSMP	FROM_TO
FT	HACC	HMAX	HMIN
HRISEFALL	INCLIB	ITL1	ITL3
ITL4	ITL6	ITL7	ITL8
ITOL	LIBINC	LIMNWRMOS	LVLTIM
MAXNODES	MAXSTEP	MAXTRAN	MAXV
NETSIZE	NGTOL	NMAXSIZE	NOCONVASSIST
NODCPOWNEG	NOLAT	NONWRMOS	NOQTRUNC
NOSWITCH	PCS	PCSSIZE	PCSPERIOD
PIVCHECK	PIVREL	PIVTOL	PSOSC
QTRUNC	RATPRINT	RELTOL	RELTRUNC
RELVAR	SAMPLE	SPLITC	STARTSMP
STEP	TIMESMP	TRTOL	TUNING
UNBOUND	VMAX	VMIN	VNTOL
WDB_IDELTA	WDB_VDELTA	WDB_NOSYNCHRO	XA

Table 11-5. Miscellaneous Simulation Control Options

AMMETER	AUTOSTOP	AUTOSTOPMODULO	CARLO_GAUSS
CPTIME	DEFAULTFALLTIME	DEFAULTRISETIME	DEFPTNOM
DSCGLOB	DSPF_LEVEL	FALL_TIME	FLOATGATE0
FLOATGATECHECK	FLOATGATERR	HIGHVOLTAGE	HIGHVTH
ICDC	ICDEV	INTERP	LICN
LOWVOLTAGE	LOWVTH	LVS_IGNORE_VARIABLE	M53
MC_IGNORE_BINNING	MMSMOOTH	MMSMOOTHEPS	NOICNODE
NOLICN	NOLTEDISC	NOMEMSTP	PARAMOPT_NOINITIAL
NOVATOPOCHK	PODEV	RANDMC	RGND
RGNDI	RISE_TIME	SIGTAIL	STATISTICAL

Table 11-5. Miscellaneous Simulation Control Options

TEMP_UNIT	TNOM	TPIEEE	ULOGIC
ZOOMTIME			

Table 11-6. Model Control Options

ACDERFUNC	ACM	ASPEC	BSIM3VER
DEFAD	DEFAS	DEFL	DEFNRD
DEFNRS	DEFPD	DEFPS	DEFW
ELDOMOS	FNLEV	GMIN	GMIN_BJT_SPICE
GMINDC	GRAMP	GENK	HIER_SCALE
KLIM	KWSCALE	IBIS_SEARCH_PATH	MAXADS
MAXL	MAXPDS	MAXW	MINADS
MINL	MINPDS	MINRACC	MINRESISTANCE
MINRVAL	MINW	MNUMER	MOD4PINS
MODMONTE	MODWL	MODWLDOT	NGATEDEF
NOACDERFUNC	NOAUTOCTYPE	NOCATMX	NOKWSCALE
NWRMOS	PGATEDEF	RAILINDUCTANCE	RAILRESISTANCE
REDUCE	RESNW	RMMINRVAL	RMOS
RSMALL	RZ	SCALE	SCALEBSIM
SCALM	SHRINK_FACTOR	SOIBACK	SPMODLEV
TMAX	TMIN	USEDEFAP	WARNING_ DEVPARAM
WARNMAXV	WL	YMFACT	ZDETECT

Table 11-7. RC Reduction Options

REDUCE_KEEP_INST	REDUCE_KEEP_NODE	REDUCE_KEEP_ OUTPUTS	REDUCE_MAX_CAP
REDUCE_MAX_IND	REDUCE_MAX_RES		

Table 11-8. Noise Analysis Options

FLICKER_NOISE	IKF2	JTHNOISE	THERMAL_NOISE
NOISE_SGNCONV	NONOISE		

Table 11-9. Simulation Display Control Options

ACSIMPROG	DCSIMPROG	ENGNOT	INGOLD
LOCAL_NOWARN	MAXTOTWARN	MAXWARN	MSGBIAS
MSGNODE	NOWARN	NUMDGT	PRINTLG
VERBOSE	WARN	WBULK	

Table 11-10. Simulation Output Control Options

ACOUT	ALTER_NOMINAL_TEXT	ALTER_SUFFIX	ASCII
ASCIIPLOT	BLK_SIZE	CAPTAB	COLLAPSE_DSPF_OUTPUT
CONTINUOUS_FFT	DEFRMSNTR	DISPLAY_CARLO	DUMP_EXTRACT
DUMP_MCINFO	EMPTY_MCHISTO	EXTCGS	EXTFILE
EXTMKSA	EXTMOD_GENWAVE	EXTRACT_EVAL_FINAL	EXTRACT_VECT_AXIS
HISTLIM	HISTO_ZERO	INPUT	INFOMC
JWDB_ACTRAN_USE_TIME	JWDB_EVENT	JWDB_EXTENSIONS	JWDB_PERCENT
KEEP_DSPF_NODE	KEEP_HMPFILE	LCAPOP	LIMPROBE
LIST	MAX_CHECKBUS	MAX_DSPF_PLOT	MC_NOMINAL_OP
MEASFILE	NEWACCT	NOASCII	NOASCIIPLOT
NOBOUND_PHASE	NODCINFOTAB	NODE	NODEFRMSNTR
NOEXTRACTCOMPLEX	NOMOD	NOOP	NOPAGE
NOSIZECHK	NOSMKMCWC	NOSTATP	NOTRC
NOTRCLIB	NOWAVECOMPLEX	NOXTABNOISE	OPALLDC
OPTYP	OUT_RESOL	OUT_SMP	OUT_STEP
PARAMETRIC_ACTRAN	POST	PRINT_ACOP	PRINTFILE_STEP
PRINTFILE_FREQ_STEP	PRINTFILE_TIME_STEP	SIMUDIV	STAT
TEMPCOUK	TIMEDIV	VBCSAT	VXPROBE
WRITE_ALTER_NETLIST			

Table 11-11. Optimizer Output Control Options

OPSELDO_ABSTRACT	OPSELDO_DETAIL	OPSELDO_DISPLAY_ GOALFITTING	OPSELDO_FORCE_ GOALFITTING
OPSELDO_JWDB_RUN	OPSELDO_NETLIST	OPSELDO_ NO_DUPLICATE	OPSELDO_ NOGOALFITTING
OPSELDO_OUTER	OPSELDO_OUTPUT	RESET_MULTIPLE_RUN	

Table 11-12. File Generation Options

AEX	ALIGNEXT	ASCII=val	COU
CSDF	DUMP_FILE_LIST	FSDB	FSDB_SINGLE_FILE
INFODEV	INFOMOD	ITRPRT	JWDB
NOAEX	NOCKRSTSAVE	NOCOU	NOIICXNAME
NOJWDB	NOPROBEOP	NSAFILE_FORMAT	OUT_ABSTOL
OUT_REDUCE	OUT_RELTOL	PROBE	PROBEOP
PROBEOP2	PROBEOPX	PSF	PSF_ALL_FILES
PSF_FULLNAME	PSF_NODEVICE_ NOISE	PSF_SCALARDC	PSF_VERSION
PSF_WRITE_ALL	PSFASCII	SAVETIME	

Table 11-13. Mathematical Algorithm Options

ANALOG	BE	BLOCKS=IEM	BLOCKS=NEWTON
CSHUNT	DCPART	DIGITAL	DPTRAN
GEAR	GNODE	GSHUNT	IEM
MAXORD	METHOD=GEAR	NEWTON	NODCPART
NODEFNEWTON	NORMOS	OSR	PSTRAN
SMOOTH	TRAP		

Table 11-14. Mixed-Mode Options

D2DMVL9BIT	DEFA2D	DEFD2A	DEFCONVMSG
DYND2ALOG	DYND2ALOG2	FS_SOLVE_AMS_NODES	FS_PARTITIONING
FS_PARTITION_ DEBUG	MIXEDSTEP	NO_FS_VA	PARTGATE_AMS_ALL
PARTVDD	PARTVDD_AMS_ALL		

Table 11-15. Other Options

CTEPREC	DCLOG	EPSO	MAXNODEORD
NODUPINSTERR	NOELDOSWITCH	NOFNSIEM	NOINIT
SEARCH	VAMAXEXP	ZCHAR	

Cadence Compatibility Options

- **SDA**

  When set to 2, this parameter enables Cadence **WSF** compatible ASCII or binary output files for EDGE and OPUS. See **WSF** and **WSFASCII** options. WSF is only supported on the Solaris platform. This option requires license authorization. Default is 0.

- **WSF**

  When **SDA**=2, Eldo creates a Cadence **WSF** binary output (*.wsf*) file for EDGE and OPUS. WSF is only supported on the Solaris platform. This option requires license authorization.

- **WSFASCII**

  When **SDA**=2, Eldo creates a Cadence **WSF** ASCII output (*.wsf*) file for EDGE and OPUS. WSF is only supported on the Solaris platform. This option requires license authorization.

SPICE Compatibility Options

- **SPI3ASC**

  Forces Eldo to create a SPICE3 compatible ASCII output (*.spi3*) file.

- **SPI3BIN**

  Forces Eldo to create a SPICE3 compatible binary output (*.spi3*) file.

- **SPI3NOCOMPLEX**

  Allows to choose which format should be used for the SPICE3 compatible binary output (*.spi3*) file header in AC analysis. By default, complex names are written (v(out)). If this option is specified, Eldo switches to pre-v6.6 behavior, that is, vr(out) and vi(out) are written.

- **SPICEDC**

  Forces Eldo to use the Berkeley SPICE mechanism when calculating the DC operating point rather than using the normal method. By using this method, VBE junctions of BJTs are set to 0.7V, while voltages across other PN junctions are set to 0V. This option can increase the efficiency on circuits containing BJT elements.

- **SPIOUT**

  DC operating point is sorted alphanumerically.

Simulator Compatibility Options

- **COMPAT**

  This is equivalent to the `-compat` flag used when invoking Eldo. Provides HSPICE compatibility. Note this option must be set at the top of the design, otherwise results can be unpredictable. Option **COMPAT** is equivalent to setting both **COMPMOD** and **COMPNET** options shown below:

- **COMPMOD**

  Triggers only the automatic conversion of models. Provides HSPICE models compatibility. This is equivalent to the `-compmod` flag used when invoking Eldo.

- **COMPNET**

  Causes the netlist to be interpreted as compatible format, but the models themselves are treated as Eldo SPICE models. This means it is assumed that models are already Eldo models. Provides HSPICE netlist compatibility. This is equivalent to the `-compnet` flag used when invoking Eldo.

Please refer to the "HSPICE Compatibility" on page 1373 for further information.

- **MOTOROLA**

  Invokes the Motorola (SSIM model) mode of Eldo. This option is equivalent to the `-ssim` command-line flag of Eldo. This model is only supported on Solaris platforms in Eldo 32-bit mode. See also "Motorola SSIM Model (Eldo Level 54 or SSIM)" on page 276.

- **USE_SPECTRE_CONSTANT**

  Allows Eldo to understand a number of Spectre default parameters, constants, and functions. See "Spectre Default Constants and Functions" on page 1398 for further information.

Netlist Parser Control Options

- **ACCSEMICOL**

  Forces Eldo to consider the semicolon character ';' in a netlist as a regular character. By default, the semicolon character ';' in a netlist is considered a space. Some netlisters exist which make use of this character for node names.

- **ADMS_FAST_PARSE**

  This option is used to accelerate the elaboration phase of the design with Questa ADMS. Please refer to Black-Box Mode for Eldo and ADiT in the *Questa ADMS User's Manual* for further information.

- **ADMSBS**

  This option enables extended identifiers to be used in VHDL-AMS descriptions. This option is used, and set by default, with Questa ADMS. It is possible to disable this by using specifying **NOADMSBS**.

- **ALTER_ADDSTEP**

  Appends **.STEP** commands found in **.ALTER** statements, instead of substituting. By default, Eldo substitutes the **.STEP** commands in the main netlist by those found in **.ALTER** statements.

- **ALTERELDO**

  Changes the way the **.ALTER** re-run feature works in compat mode. In compat mode, for alter index 'n', Eldo revisits the 'n-1' alter looking for substitution. In default Eldo mode, for alter index 'n', Eldo only deals with the nominal and the alter 'n'; it ignores the 'n-1' alter. In other words, in compat mode, alters are cumulative, but not in Eldo default mode. Specify **ALTERELDO** to activate the default Eldo mode behavior when in compat mode.

- **ALTINC**

  Forces Eldo to replace the first **.INCLUDE** statement found in an input netlist by the first **.INCLUDE** statement found in the **.ALTER** section of the netlist. This allows the replacement of one file by another one when using the Eldo re-run facility.

- **BSLASHCONT**

  Allows two backslashes to be used for the continuation of the line. Example:

  ```
  .option BSLASHCONT
  R1 1 2 \\
  3k
  R2 1 2 1k
  ```

  R1 will be set to 3K.

- **CHECKDUPL**

  Forces Eldo to perform the check of duplicate instance names, even when the netlist is larger than the internal limit (1000 lines). This option is position-dependent, it takes affect only after it has been set in the netlist, if the option is set at the end of the netlist it will have no effect.

- **CKDCPATH**

  Forces Eldo to issue a warning instead of an error when a dangling node (no DC path to ground) is encountered on a current source. The source is then disabled and the node connected to ground. This option is only used in -compat mode.

- **COMPEXUP**

  Forces Eldo to keep the name of the extraction results (from **.EXTRACT** and **.MEAS** statements) in uppercase when in simulator compatibility (`-compat`) mode. Prior to Eldo v6.6 the names were always in uppercase, beginning Eldo v6.6 they are in lowercase.

- **CONTINUE_INCLUDE**

  Specifies that continuation lines with + as the first character apply to the **.INCLUDE** command in the file. For example, if the main netlist has the two lines:

  ```
  .include file.inc
  + b=2
  ```

  With this option specified, if file *file.inc* contains as its last line:

  ```
  .param a=1
  ```

  Eldo would interpret this as:

  ```
  .param a=1
  + b=2
  ```

- **DICPRIO**

  Forces **.IC** statements on nodes to have priority over any coming from a **.USE** command. By default, if **.USE** file_name **IC** is specified in the netlist, and there are also some **.IC** statements on nodes, then nodes emulated via the **.USE** command have higher priority than those on **.IC** statements.

- **DOTNODE**

  Forces Eldo to accept non-hierarchical characters in node names. Must be set if you want to have node *a.0* created as a regular node. For backward compatibility. In versions up to 2006.x, Eldo accepted a non-hierarchical node name containing the hierarchical character (usually '.'), providing that the node name did not begin with an X. For instance, *foo.1* was accepted as a node name, but *X1.1* was recognized as a hierarchical node and therefore Eldo expected a local node named "1" in subckt *X1*. However because ADMS subcircuit instances do not necessarily begin with an X, the possibility of having the hierarchical character in node name is not allowed in ADMS, which can lead to incompatibility between Eldo and ADMS. The hierarchical character in non-hierarchical names is not allowed (beginning 2007.1), so that Eldo and ADMS conventions match. In ADMS, this option will be ignored.

- **ERR0DIV0**=0|1

  When set to 1, this forces Eldo to return an error for a zero divided by zero calculation. Default is 0. By default, 0/0 returns zero.

- **EXTERR**

  Forces Eldo to stop and generate errors when there are problems found in **.EXTRACT** or **.MEAS** commands. By default, Eldo only generates warnings when errors are found in **.EXTRACT** or **.MEAS** commands. Such statements will be ignored. Prior to Eldo v6.7 errors were always generated.

- **KEEPDANGLING**

  Maintains dangling objects in the Eldo database. By default, Eldo disregards dangling objects and shorted MOS, diodes, resistors and capacitors. This can create problems with DSPF because not all objects might be retrieved. Dangling and shorted objects could be the result of dummy devices added to improve device matching. Parasitics of such devices are required.

- **KEEPSHORTED**

  Maintains shorted objects in the Eldo database. By default, Eldo disregards dangling objects and shorted MOS, diodes, resistors and capacitors (objects for which all pins are the same). This can create problems with DSPF because not all objects might be retrieved. Dangling and shorted objects could be the result of dummy devices added to improve device matching. Parasitics of such devices are required.

- **LOOPV0**

  By default, Eldo issues an error (27) when it detects a voltage loop. Use this option to force Eldo to allow a voltage loop or inductor loop of a zero voltage source. Such loops could be generated from an automatic circuit extraction. When specified, any attempt to change the value of one of the sources in a loop will result in an Eldo error. See option **VOLTAGE_LOOP_SEVERITY** to downgrade a voltage loop error to a warning.

  Rules on Eldo reporting of voltage loops:

  - For a single device with two pins shorted:

    By default, for a single device with two pins shorted, Eldo issues an error when one of the following conditions is met:

    - the source value is positive, for example:

      ```
      v1 n1 n2 10
      ```

    - the source value depends upon a parameter, for example:

      ```
      v1 n1 n2  p1
      ```

    - the source has a total loop voltage of 0, for example:

      ```
      v1 n1 n2   0
      ```

      No error is issued for this scenario if option **LOOPV0** is set.

    In compat mode (**-compat** flag or option **COMPAT**), no error will be issued.

  - For two or more inductors or voltage sources connected in such a way they form a loop:

    - To downgrade an error to a warning, specify option **VOLTAGE_LOOP_SEVERITY**= **WARNING**. Default is **ERROR**.

    - If all the source values in the loop are zero then use option **LOOPV0** to disable the checks as in the case of a single device, for example:

      ```
      v1 n1 n2 0
      v2 n1 n2 0
      ```

    - If one of the voltage loop values is different from zero an error will be issued unless option **VOLTAGE_LOOP_SEVERITY**= **WARNING** is set, for example:

```
v1 n1 n2 0
v2 n1 n2 1
```

In this example the checks cannot be disabled by option **LOOPV0**.

In compat mode (**-compat** flag or option **COMPAT**), you *cannot* override the error (option **LOOPV0**).

- **MACMOD**=[1|2|3|0]

  Extended MOSFET element support. Enables MOSFET elements access to subcircuit definitions or Verilog-A models when there is no when there is no corresponding model reference on the **.MODEL** statement. It also enables a subcircuit (X) instance access to a model reference when there is no corresponding subcircuit definition or Verilog-A module. Option **MACMOD** is equivalent to option **MACMOD**=1.

  In other words, if Mxxx is in a nelist this could be a subckt instance even if the name begins with M. If Xxxx is in a netlist it could be a MOSFET element, a subckt instance, or a Verilog-A module instance.

  When a **.HDL** command (see “.HDL” on page 678) is used to compile a Verilog-A model then it will be instantiated by an X-instance. When **MACMOD** is enabled then the X-instance could be a MOSFET.

  MACMOD=1
  For a MOSFET element Mxxx of model name MNAME Eldo will first look for a MOS **.MODEL** statement with the name MNAME. If a MOS model MNAME is not found Eldo will look for a subckt definition of the name MNAME. If a subckt definition is found Eldo will use it for the Mxxx instance. The number of terminals must match.

  MACMOD=2
  For an X-element (subckt instance or Verilog-A instance) Eldo will first look for a subckt definition or a Verilog-A module. If a subckt definition or a Verilog-A module is not found Eldo will look for a MOS model definition.

  MACMOD=3
  Enables both of the above features.

  MACMOD=0
  Disables the features. Default.

- **MEAS_TARGWHEN**

  This option affects the interpretation of the **TARG** specification on **.MEAS** commands using **WHEN** statements. By default, Eldo ignores the **TARG** specification. Specify this option to return to the pre-2009.2 behavior for a warning to be generated if **TARG** is used in **WHEN**-type **.MEAS** statements.

- **MTHREAD**

  Activates multi-threading for a single DC or TRAN simulation. Eldo will share computer resources on a multi-processor machine. Eldo will make use of all the possible CPUs on the machine. This option is equivalent to the Eldo `-mthread` command line flag.

See “Multi-Threading Eldo Simulations” on page 58 for a full description of multi-threading in Eldo.

- **NOACT0**

  Forces AC calculation only at the specified timevalues for a combination of **.AC** with **.OP** timevalues. Default behavior (without the option) is that an AC analysis is performed at time 0.

- **NOALTINCEX**

  By default the content of **.INCLUDE** commands specified in **.ALTER** blocks is treated by Eldo as if the content is written at full length in the netlist and substitutes accordingly. In previous versions (pre-v6.9), the **.INCLUDE** commands specified in **.ALTER** blocks were added (extended) to the nominal circuit, for example:

```
i1 1 0 1
r1 1 0 1
.op
.extract dc v(1)
.alter
.include altincex.a1
.end
```

  assuming file *altincex.a1* contains the line r1 1 0 2, for the first ALTER run Eldo will substitute r1 1 0 1 by r1 1 0 2 in the netlist (behaves as if line r1 1 0 2 was in the netlist). For Eldo versions before v6.9, Eldo assumed two resistors in parallel to the I1 device. For backward compatibility, specify option **NOALTINCEX** to switch back to the old mechanism.

- **NOADMSBS**

  Disables option **ADMSBS** which is set by default with Questa ADMS. With **NOADMSBS** specified, extended identifiers cannot be used in VHDL-AMS descriptions.

- **NOBSLASHCONT**

  Disables option **SLASHCONT** when invoking Eldo with the `-compat` flag.

- **NOCMPUNIX**

  The characters “[” and “]” are interpreted as special characters when the wildcard character “*” is used. According to Unix convention, these special characters may be overridden by using the backslash character “\” which removes the special function of the character that immediately follows it.

- **NOELDOLOGIC**

  Specifies that subcircuit X instances can be named beginning with any Eldo logic or macro primitive name. With this option, Eldo will not attempt to parse such instances as logic gates. This option is automatically set when invoking Eldo with the `-compat` flag. This option affects the following Eldo built-in logic primitives: AND, OR, NAND, NOR, XOR, CMP, CMPD, INV; and the following Eldo built-in macromodels: ADC, DAC, DEL, FNS, FNZ, OPA.

- **NOERR_XPINSMISMATCH**[=subckt_name]

  When this option is set, if the X instance contains more pins than the SUBCKT definition, Eldo will allow this but only the first pins in the SUBCKT will be taken into account.

  The mismatch in the number of pins will not generate warning/error messages if option **NOERR_XPINSMISMATCH** is specified without any argument, or if the subckt name matches that specified in the command.

  This enhancement works only for Eldo, not ADMS.

- **NOMATSING**

  This option can be used to complete a simulation if the simulation stops with a warning regarding a singularity. Otherwise, it will be stopped if a singularity is found in DC. However, the design has to be improved if there is a singularity. Even if the simulation continues, this does not mean that the solution in DC is correct.

  Please refer to the appendix "Improved Diagnostics for Certain Erroneous Models" in the *Questa ADMS User's Manual* for further information.

- **NOSETBUSEXPAND**

  Disables automatic bus expansion inside the .SETBUS command.

- **NOSSTKEYWORD|**
  NOKEYWPARAMSST

  Tells the parser to bypass the RF keyword check in expressions or in **.PARAM**. For a full list of Eldo RF keywords that cannot be used in **.PARAM**, see "Reserved Keywords Not Available in .PARAM if an RF Analysis is Specified in the Netlist" on page 779.

- **NOZSINXX**

  Remove the effect of the *sinx/x* term used in the AC response of Z transforms. The response was modified in Eldo v6.3, previously the term was not taken into account.

- **PARAM_BEFORE_USE**[=0|1]

  When set to 1, parameters must be defined before being used. Default is 0. Note that option **STRICT** sets the value of **PARAM_BEFORE_USE** to 1; it is possible to reset **PARAM_BEFORE_USE** to 0 after option **STRICT** has been specified. For example:

  ```
  .option STRICT
  .OPTION PARAM_BEFORE_USE = 0
  ```

  allows a parameter to be used before its definition, such as in:

  ```
  .PARAM p2 = p1
  .param p1 = 1
  ```

- **PARHIER**='local'|'global'|'hier'|'hierlocal'

  This parameter controls the priorities for parameters. Quotes are optional. Default for Eldo is **parhier**=hier (was local in pre-v6.6 versions of Eldo). When -compat is set, default is global.

When PARHIER is GLOBAL or LOCAL, then the parameter priorities adopted in X statements are performed as if the operations were executed inside the SUBCKT. The following rules apply:

When PARHIER is GLOBAL, the priority is the following:

1. .PARAM statement at a higher level
2. Value at the X call (instance)
3. SUBCKT definition

When PARHIER is LOCAL, the priority is:

1. SUBCKT call (instance)
2. SUBCKT definition
3. .PARAM statement

When PARHIER is set to HIER or HIERLOCAL, then the mechanism for evaluating parameters mimics that of the function call in programming languages such as C. The parameters found on the right-hand side of the expressions are sought first in the current environment, while the name on the left-hand side refers to the parameter name inside the subcircuit being called. This is a very different mechanism than for GLOBAL or LOCAL. For example:

```
ID  VDD 0 1m
RC1 S1 VDD 1k
X1 S1 0 sub1

.op

.subckt sub1 c b
.param w = 1k
XM1 c b  sub2 w2 = w
.ends sub1

.subckt sub2 c b
.param w = 2k
r1 c b 'w2'
.ends sub2
.param w = 7k
.end
```

If PARHIER = GLOBAL w2 is 7k. Because it is as if we had:

```
XM1 c b  sub2 xm1.w2 = xm1.w
```

and xm1.w takes the definition at the highest level.

If PARHIER = LOCAL w2 is 2k. Because it is as if we had:

```
XM1 c b  sub2 xm1.w2 = xm1.w
```

and xm1.w takes the definition at the lowest level, i.e inside the subcircuit SUB2.

If PARHIER = HIER w2 is 1k. Because it is as if we had:

```
XM1 c b  sub2 xm1.w2 = w
```

and local value for w is 1k.

If PARHIER = HIERLOCAL w2 is 1k: same as PARHIER=HIER.

However, in case of PARHIER = HIER or PARHIER = HIERLOCAL, if a parameter is specified at the X instance, then this value will have precedence over the local value if any. For example:

```
ID  VDD 0 1m
RC1 S1 VDD 1k
X1 S1 0 sub1

.op

.subckt sub1 c b
.param w = 1k
XM1 c b  sub2 w=3k w2 = w
.ends sub1

.subckt sub2 c b
.param w = 2k
r1 c b 'w2'
.ends sub2
.param w = 7k
.end
```

Here, Eldo will assume that the w on the w2 =w statement is 3k because w is explicitly specified on the X instance. If w=3k had not been there, then the local value 1k would have been taken.

Note that in case a parameter which appears on the right-hand-side of an expression on a X instance has both a value in the current environment, and in the SUBCKT being called, and is not specified on that X instance, then Eldo will issue a warning that the two values exist, just to warn the user about possible problems in the netlist.

The difference between HIER and HIERLOCAL is for the case a parameter which appears on the right-hand-side of an expression on a X instance, is neither specified on the same X call, neither specified on the current environment: if PARHIER is set to HIER, Eldo will issue an error that the parameter is not found, if PARHIER is set to HIERLOCAL, then Eldo will check for this parameter inside the SUBCKT being called, and will use that value if found. For example:

```
ID  VDD 0 1m
RC1 S1 VDD 1k
X1 S1 0 sub1

.op

.subckt sub1 c b
XM1 c b  sub2 w2 = w
.ends sub1
```

```
.subckt sub2 c b
.param w = 4k
r1 c b 'w2'
.ends sub2
```

If PARHIER is set to HIER, Eldo will issue an error while evaluating w2 = w, because w is not found in the current environment. If PARHIER is set to HIERLOCAL, Eldo will check inside the sub2 if w exists, and will find it.

- **PATTERN_MAX_ALLOWED_COEFF**

  Changes the limit of the number of bits used in the alternative syntax for a Pattern Function specification. Default is 100.

- **PEVFLY**

  Evaluate implicit parameter expressions "on the fly" instead of storing them in memory before evaluation. This can save memory.

  In case a circuit contains many implicit expressions, for example:

  ```
  V1 1 0 PWL ( {2*p1} {p2} {3*p1} {p2} {4*p1} ... }
  ```

  where the number of couples (time-value, input-value) can be as high as several millions, then the amount of memory used by Eldo to store the implicit expressions can be huge. If option PEVFLY is set, then implicit expressions which are specified at the top level (not in a .SUBCKT) and after the .OPTION, will be evaluated immediately. This will save a lot of memory. Note that this option has some side effects:

  - double definition of parameters is not allowed
  - a parameter must be defined before it is used in the implicit expressions
  - The dependency tree of the implicit expression is not stored: therefore if a parameter is changed (for example via a .STEP command) then implicit expressions which have been optimized by the option and which depend on this parameter will not be re-evaluated, and results might not be as expected

  Eldo will issue an error if it falls into the first two cases above. For the third case, a warning will be issued if some parameter values change at runtime (.step for instance).

- **POWNEG0**

  If an expression exists of the form *a****b* (a^b), with *a* being negative and *b* being a non-integer value, by default Eldo issues an error. With option **POWNEG0**, the expression will return 0.0.

- **QUOTREL**

  This option instructs Eldo to consider double quotes as single quotes. This provides improved compatibility with other simulators. However, character strings are not recognized as such in **.PARAM** statements. This option is set by default in compat mode (**-compat** flag or option **COMPAT**).

- **QUOTSTR**

  This option instructs Eldo to consider double quotes as a parameter string delimiter. This is the default in Eldo, but disabled in compat mode (**-compat** flag or option **COMPAT**). Therefore, only use this option in compat mode to identify parameter strings with double quotes.

Note

If both **QUOTREL** and **QUOTSTR** options are specified, the last to be specified is used.

- **RMV0**

  Forces Eldo to connect together the two pins of zero voltage sources if they are not used as an ammeter (that is, if the current flowing through it is never used as a command to control other devices). Connecting the two nodes simplifies the system to be solved.

- **SLASHCONT**

  Allows a single backslash to be used for the continuation of the line. This is automatically set when invoking Eldo with the `-compat` flag. Example:

```
.option SLASHCONT
R1 1 2 \
3k
R2 1 2 1k
```

  R1 will be set to 3K. When invoking Eldo with the -compat flag, it is possible to disable this option by using **.OPTION NOBSLASHCONT**.

- **STOPONFIRSTERROR**=1|2

  When set to 1, Eldo will stop parsing the netlist on the first error. This is to prevent the case where Eldo cannot recover correctly from a first error, and would display many subsequent meaningless errors. If the netlist file contains **.ALTER** statements for multiple simulation runs, Eldo will stop on the first **.ALTER** that has an error, but continue with the remaining **.ALTER** statements.
  When set to 2, the first error stops the simulation even if the netlist file contains **.ALTER** statements for multiple simulation runs.

- **STRICT**

  Eldo accepts parameter names that contain special characters '!' '|' '<' '>' '?' '&' (in addition to letters, numbers and characters '_' '$' '[' ']' '@' '\' '~' ':' '#' and '%'). If option **STRICT** is set, Eldo will issue an error when using such special characters, and also in the following cases:

  - missing ' or { around expressions
  - double definition of parameter in **.PARAM** statements
  - use of parameter before definition
  - double definition of parameters in **.MODEL** cards
  - use of unknown parameter for a model

- double definition of object

Option **STRICT** also sets the value of **PARAM_BEFORE_USE** to 1, which imposes that parameters must be defined before being used. See **PARAM_BEFORE_USE** for more information.

- **SUBALEV**

When this option is set with **.ALTER** statements in the netlist, then Eldo will substitute all matching lines including those inside subcircuits being substituted. By default (beginning Eldo v6.8) Eldo will substitute only those lines at the top level, to be compatible with other simulators. This option is provided to switch back to the previous behavior for backward compatibility.

- **SUBFLAGPAR**

When this option is set, then **ANALOG** and **ELDO** are no longer considered as keywords, and these names will be allowed in the subcircuit definition/instantiation of a subcircuit. This option must be specified at the top of the netlist, just after the title line.

By default, it is not possible to create and instantiate a subcircuit named **ANALOG** or **ELDO**. Therefore, the following design would fail:

```
Title
.subckt analog a b
r1 a b 1
.ends
x1 a b analog
v1 a 0 1
rb b 0 1
.dc
.end
```

However, if option **SUBFLAGPAR** is set, then it will be allowed. In such a case, and if the user also wants to set the subcircuit as ANALOG (that is, that the subcircuit should be solved using Newton block iteration techniques), then that flag must be enclosed in brackets, for example:

```
.OPTION SUBFLAGPAR
x1 a b analog (analog)
```

The first occurrence of the parameter `analog` in the above example stands for the subcircuit name, while the second occurrence, enclosed in brackets, stands for the “high accuracy” flag **ANALOG**.

- **USE_LOCATION_MAP**=filename

Specifies the full path and filename to the location map file. Location maps are used to replace prefixes of physical pathnames with environment variables (soft pathnames). If the filename is not found, Eldo displays a warning message.

See “Location Maps” on page 61 for further information.

- **USEFIRSTDEF**

  Forces Eldo to only use the first definition of **.MODEL**, **.SUBCKT**, or **.PARAM** statements. Any further definitions are ignored, to allow the simulation to proceed. By default, when Eldo encounters multiple definitions of **.MODEL** or **.SUBCKT** statements Eldo reports an error, and stops the simulation. By default, subsequent definitions of parameters (with **.PARAM**) do not cause an error, but instead overwrite the previous definition.

  Limitation: if this option is used with **.ALTER**, then new parameter values which appear in the **.ALTER** section will be ignored, since the first value will always be used.

- **USETHREAD**=VAL

  Activates multi-threading for a single DC or TRAN simulation. Eldo will share computer resources on a multi-processor machine. Eldo will make use of the number of CPUs as specified with this option. The number specified can exceed the number of CPUs available, but this is not recommended. This option is equivalent to the Eldo `-usethread val` command line flag.

  See “Multi-Threading Eldo Simulations” on page 58 for a full description of multi-threading in Eldo.

- **VOLTAGE_LOOP_SEVERITY**=**ERROR|WARNING**

  By default Eldo issues an error (27) when it detects a voltage loop. Use this option to force Eldo to generate a warning instead of an error. Default is **ERROR**. See option **LOOPV0** to allow a voltage loop or inductor loop of a zero voltage source, and for more details on voltage loop handling in Eldo. Not supported in compat mode.

- **WARN2ERR**=VAL

  Raises the level of a warning into an error. Eldo will then stop the parsing when this “warning” is reported. Specify as VAL the warning number you want raised. This option must be placed at the beginning of the netlist.

- **XBYNAME**

  Map subcircuit pins or nets by name, not by position (default). Allows a subcircuit (X call) to be instantiated using pin/net names of a subcircuit definition (**.SUBCKT** command).

  When this option is specified, it is possible to mix both syntaxes in the same netlist, Eldo will check for the presence of at least one of the keywords **PIN:**, **PARAM:**, **NET:**, **MODEL:** or **KEYWORD:** to select which syntax to be used for each X instance.

  See also Subcircuit Instance.

Simulation Speed, Accuracy and Efficiency Options

Before using the following options it is recommended that the relevant section be consulted in the Speed and Accuracy chapter.

- **ABSTOL**=VAL

  Absolute current accuracy. The default value is **VNTOL** × **ITOL**. Note **ABSI** is synonymous to **ABSTOL**.

- **ABSVAR**=VAL

  Used for timestep control. Sets the maximum voltage change from convergent iteration to iteration (timestep to timestep) for the condition that **LVLTIM**=1 and **DVDT**=0. If the simulator produces a convergent solution that is greater than **ABSVAR**, the timestep is reduced for the next solution. Default is 0.2 V.

- **ADJSTEPTRAN**

  When this option is set the TPRINT parameter of the .TRAN command is used to calculate minimum and maximum time step values. They are calculated as follows:

  MIN = TPRINT/10
  MAX = MIN(TIME_DURATION/50, TPRINT×RMAX)

  Where TIME_DURATION is the time duration specified in the **.TRAN** command and RMAX is the value given by the option **RATPRINT**. When **RATPRINT** is not specified and **ADJSTEPTRAN** is active then **RATPRINT** defaults to 2.

- **AIDSTP**

  Forces Eldo to use convergence aid for all **.STEP** simulations.

  When convergence aid is present in the netlist (option **GRAMP**, **.RAMP TRAN**, and so on), in combination with **.STEP**, then the convergence aid instructions are taken into account only for the first step. The assumption being that since subsequent runs use the results of previous **.STEP** simulations to converge, the convergence aid will not be needed. However, sometimes this assumption is not true.

- **CAPANW**=VAL

  Value below which coupling capacitors could be treated by OSR. Defaults to 50 femto Farads, unless option **DIGITAL** is used, in which case it is 1 nF.

- **CHGTOL**=VAL

  Is the absolute tolerance on charge and is used by charge control devices, (for example BIP, DIODE, JFET, BSIM1). Default value is set by **EPS**. This option is only used by the “Gear” algorithm or if option **QTRUNC** is set.

- **DVDT**=VAL

  Used for timestep control on Newton blocks. When set to 0, the **DVDT** algorithm is activated for the condition that **LVLTIM**=1. Default is –1.

- **EPS**=VAL

  Sets the internal simulator accuracy. The default value of **EPS** depends on the highest voltage levels found in the circuit, based upon the analysis of independent voltage sources. If all voltage levels are below 1.9V, then the default **EPS** is set to 1mV. If any voltage level is higher than 1.9V, then the default **EPS** is set to 5mV. Explicitly specifying an **EPS** value, **.OPTION EPS**=1e-5 for example, overrides this rule and sets **EPS** to the specified value (1e-5 in this example), regardless of the voltage sources present in the circuit. This voltage source rule is also ignored if **EPS** is set indirectly via **.OPTION TUNING=**, and the value for **EPS** depends only on the **TUNING** setting, see "Global Tuning of the Accuracy—EPS" on page 1070.

  Values smaller than $1.0{\times}10^{-10}$ V need to be defined with the **UNBOUND** parameter (see "UNBOUND" on page 973).

- **FASTRLC**

  For R, L and C elements which have values that vary at run time, a new device evaluation is performed at each iteration to ensure accurate results. When **FASTRLC** is set, the value used at a given time is not recomputed at each iteration. Instead, the value of previous time step is used. This accelerates the simulation because expressions are not re-evaluated at each time step, at the cost of a lower accuracy. This option only has an effect in TRANSIENT simulation.

- **FLUXTOL**=VAL

  Is the absolute tolerance on flux (for inductors). Default value is set by **EPS**. This option is only used by the "Gear" algorithm or if option **QTRUNC** is set.

- **FREQSMP**=VAL|{VAL1,VAL2,...VALn}

  Forces Eldo to compute a time point at every multiple time interval of 1/**FREQSMP**, the sampling frequency. Useful when performing a Fourier analysis. Multiple values can be specified as a list in which case Eldo computes timepoints corresponding to all sampled points. For example: a design has two clocks at 2MHz and 320 kHz. For uniform sampling, time intervals of 0.5µs and 3.125µs are needed to calculate the necessary exact points (for FFT post processing). Note that if options which impose the sampling frequency are set (for example interpolate=0 on **.OPTFOUR**), the freqsmp array is reset to a single value. It is possible to assign expression parameters to **FREQSMP**, for example:

  ```
  .PARAM p1=1k
  .OPTION FREQSMP=p1
  ```

- **FROM_TO**=0|1

  When enabled (=1), this option will affect the functions min, max and avg of the **.MEAS** command. The simulation will stop as soon as all measurements other than min, max and avg which don't have a from= to= parameter are complete. Though the results of these measurements will be not be complete, the simulation will run faster. The default value is 0.

- **FT**=VAL

  Used for timestep control. When a solution is rejected because of non-convergence (in conjunction with **ITL3** and **ITL4**) the timestep is reduced (multiplied) by the specified fraction **FT**. Default value is 0.125.

- **HACC**=VAL

  Sets the acceleration factor for timestep control. Default value is 2, which means that Eldo will attempt to multiply the current timestep by 2 at most. This time step acceleration strategy is a compromise and is a robust choice providing the best results on average. You are discouraged from changing the value of this option. See “More about time step control” on page 1068 for further details.

- **HMIN**=VAL

  Sets the minimal internal timestep. Default value is 1 ps, which is a value well suited for typical MOS circuits. The default for Switched Capacitor circuits is 2 ns. The default for bipolar circuits is 10 ps. See “More about time step control” on page 1068 for further details.

- **HMAX**=VAL

  Sets the maximum internal timestep. Default **HMAX** value is 1/10 of the wave period when using **SIN** and **SFFM** functions. See also **.TRAN** TPRINT TSTOP [TSTART [HMAX]] [**UIC**] in “.TRAN” on page 911. See “More about time step control” on page 1068 for further details.

- **HRISEFALL**=VAL

  Forces Eldo to take timesteps not greater than *dt/val* during transitions as defined by PULSE or PWL signals. VAL is an integer. *dt* is the transition time between two time-points corresponding to different values. Default value is 0. Can be specified by a parameter or expression. For example:

  ```
  .option HRISEFALL = p1
  .param p1 = value
  ```

- **INCLIB**

  Same as the **LIBINC** option. Specifies that the full contents of every library are read by Eldo in one pass upon completion of reading the input file. This was the default mechanism in the v5.8 version of Eldo. All the libraries (.LIB) are included without filtering the objects (model, card, or subcircuit) that are not used in the specific netlist.

- **ITL1**=VAL

  Sets a limit on the maximum number of DC iterations. Default value is 100.

- **ITL3**=VAL

  Used for timestep control. If convergence is reached in less than **ITL3** iterations, the next timestep is doubled. Default value is 3.0.

- **ITL4**=VAL

  Used for timestep control. If convergence is not reached within **ITL4** iterations, the present timestep is rejected and the next one reduced by **FT**. Default is 13.0.

- **ITL6**

  Equivalent to **ITL3** when DC convergence assist is in use. Default value is 5; however this default is 6 in Pseudo-Tran algorithm.

- **ITL7**

  Equivalent to **ITL4** when DC convergence assist is in use. Default value is 30; however this default is 20 in Pseudo-Tran algorithm.

- **ITL8**

  This controls the maximum number of iterations allowed for each trial of ramping algorithm (**GRAMP**, **PSTRAN**, **.RAMP DC**). Default is 10000.

- **ITOL**=VAL

  Controls the current accuracy of the simulator when solving circuits using Newton iterations. Circuit convergence is reached when:

  $$I(i) - I(i-1)| < RELTOL \times |max(|I(i)|, |I(i-1)|)| + VNTOL \times ITOL$$

  where $I(i)$ is the current value at voltage iteration i and $I(i-1)$ is the previous iteration. Default value is 1.0×10^{-6} A/V.

- **LIBINC**

  Same as the **INCLIB** option. Specifies that the full contents of every library are read by Eldo in one pass upon completion of reading the input file. This was the default mechanism in the v5.8 version of Eldo. All the libraries (**.LIB**) are included without filtering the objects (model, card, or subcircuit) that are not used in the specific netlist.

  This mode is also activated in the case of option **COMPAT**. Option **LIBINC** can also be activated by using the command line flag `-libinc` at invocation of Eldo.

- **LIMNWRMOS**=VAL

  This option is used to set the value below which MOS resistors are not handled at all. They are not created as objects and not handled in the approximative manner described in the **NONWRMOS** option.

  This option is used to collapse intrinsic MOS transistor nodes. With this option, the effect of the parasitic resistors upon the channel current is still taken into account, although not as accurately as when the nodes are explicitly created. Particularly, some of the overlap capacitances in the MOS model become connected to the external drain/source, whereas they really are connected to the internal nodes. This may change results slightly.

- **LVLTIM**=VAL

  Sets the simulator Time Step Control algorithm. See also “Time Step Control—Algorithm Selection with the LVLTIM Option” on page 1066.

- **MAXNODES**=VAL

  To optimize memory allocation when simulating large circuits, the maximum number of circuit nodes may be specified. We recommend use of a value larger than the number of nodes in the circuit, to prevent memory re-allocation procedures.

- **MAXSTEP**=VAL

  Controls the maximum number of runs allowed for a **.STEP** simulation. Default value is 25000. An error message is generated when more runs are requested as follows (example):

```
ERROR 1580:  COMMAND .STEP TGIG : suspicious number of runs: 226250001.
+    The maximum allowed value can be set by .OPTION MAXSTEP=val.
+    Current value is 25000.
```

- **MAXTRAN**=VAL

  Specifies the maximum number of circuit components in order to optimize memory allocation when simulating large circuits. We recommend a value larger than the number of components in the circuit, to prevent memory re-allocation procedures.

- **MAXV**=VAL

  Sets maximum voltage values for which Eldo searches for the DC operating point of a circuit. The options **VMIN** and **VMAX** also set bounds on DC operating point voltages, but **MAXV** overrides **VMAX**. Default is 1.0×10^{13}V. Thus **MAXV** must be specified if there are operating points greater than 1.0×10^{13}V in the circuit.

- **NETSIZE**=VAL

  Provides a rough guess about the size of the Newton matrix.

- **NGTOL**=VAL

  Is the absolute tolerance on voltage. Default value is set by **EPS**. This option is only used by the “Gear” algorithm or if option **QTRUNC** is set.

- **NMAXSIZE**=VAL

  Sets the maximum number of nodes that can be contained in a single Newton block. Default value is 6.0×10^{4}. However, if the **NEWTON** option is explicitly specified, it has priority, and **NMAXSIZE** defaults to 5.0×10^{6}. This is in order for very large MOS circuits, containing over 60,000 nodes, to be simulated by default with OSR.

  The **NMAXSIZE** option works in the following way:

  Eldo attempts to partition:
  If `nb_nodes < ` **NMAXSIZE**, Eldo can attempt only one block, or it will attempt several blocks. This depends upon **EPS** and the option **BLOCK**=<> specification.
  Else, Eldo will attempt several blocks.

  However, in the process of creating blocks, **NMAXSIZE** is not checked. The matrix grows depending on the connectivity, and the matrix size may exceed **NMAXSIZE**. The growth of the matrix cannot be stopped at that step, since the simulation would probably not work if we decide to split the matrix at those points where Eldo sees potential important feedback.

Once the blocks are completed, Eldo will attempt to concatenate blocks in case it sees feedback between blocks. However, it will not concatenate blocks if the sum of their size exceeds **NMAXSIZE**.

- **NOCONVASSIST**

  Disables the automatic convergence aid mechanisms. This is specified to avoid other algorithms when non-convergence occurs. In this case Eldo will try the first algorithm and if there is a non-convergence problem the simulation is stopped and Eldo returns the list of nodes.

- **NODCPOWNEG**

  The power dissipation returned at DC operating point takes into account the power of all independent V and I sources (beginning v6.8). Specify this option for backward compatibility; only those V and I sources which generate power are considered (that is, those for which I×V are negative), which corresponds to the majority of the cases.

- **NOLAT**

  Suppresses latency optimizations.

- **NONWRMOS**

  This option is used to collapse intrinsic MOS transistor nodes. With this option, the effect of the parasitic resistors upon the channel current is still taken into account, although not as accurately as when the nodes are explicitly created. Particularly, some of the overlap capacitances in the MOS model become connected to the external drain/source, whereas they really are connected to the internal nodes. This may change results slightly.

  When this option is active, there might be situations where DC cannot be found. If this happens, remove the option. Even when DC is found, if the netlist contains some MOS devices with forward biased bulk-source and bulk-drain junctions, Eldo will detect such situations during DC convergence, and will resimulate with the option disabled.

  See also "Collapse the intrinsic MOS transistor nodes" on page 1077 for further information.

- **NOQTRUNC**

  Disables the charge-based *Local Truncation Error* algorithm. See option **QTRUNC**.

- **NOSWITCH**

  Forces Eldo to ignore the **SWITCH** keyword specified as part of a subcircuit instance in the netlist.

- **PCS**=0|1|2|3

  (Periodic Circuit Speedup) Used to increase the simulation speed for circuits with periodic or nearly periodic nature. PLLs in near-lock state belong to this category. The amount of speedup depends on the design nature. The speedup will be more significant with relatively large circuits. With circuits showing no periodicity at all, the option will not usually provide any speedup and may even slow down the simulation. Periodic Circuit Speedup is invoked by setting **PCS**=1|2 for BSIM3v3 models only (**PCS**=1|2 represent two possible speed optimization methods, **PCS**=1 is more recommended) or **PCS**=3 for BSIM4 and BSIM3v3 models (with same speed optimization as **PCS**=1). Default is 0. This option only supports the BSIM3v3 and BSIM4 models.

Note

BSIM4, unlike BSIM3, has many parasitic configurations: gate resistors, body resistors network, bias dependent access resistors, and so on. These parasitic configurations are controlled by a set of instance and model parameters (Rdsmod, Rbodymod and Rgatemod). As the complexity of the parasitic configuration around the core model increases, the gain expected by the PCS decreases.

Caution

Use of the PCS option in the case of non-periodic conditions may even slowdown the simulation a little, which is why this option is not set by default. In this case, try using the PCS option in conjunction with the .OPTPWL or .OPTWIND commands to specify the time after which to turn on PCS.

- **PCSSIZE**=VAL

  This option is used to specify the memory size (in MB) to be used by the PCS algorithm. The default value is 128. In many cases increasing this value will enhance the speedup achieved by the PCS option. However this value should not exceed the available physical memory to prevent memory allocation problems and excessive slowdown due to disk access.

- **PCSPERIOD**=VAL

  This option can be used to specify a guess for the period of the circuit. When provided, this guess can be used to enhance the speedup achieved by the PCS option.

- **PIVCHECK**=1|2

  Used in the LU-Factorization algorithm in order to check the magnitude of terms in the matrix. Matrix manipulations can lead to very large numbers which may overshoot the machine's capacity to store a real value resulting in an Error Code 6. When this option is set, Eldo performs preliminary checks on matrix terms to prevent such an error. As it is time consuming it is not set by default and should not be set unless requested by Eldo. Two levels of safety are provided to prevent such errors in the matrix resolution, each one slower than the default and level 2 slower than level 1. **PIVCHECK**=1 should be specified first and if this still fails to resolve the issue then Eldo will inform you to set **PIVCHECK**=2.

- **PIVREL**

  Used in the LU-Factorization algorithm in order to find a compromise between the value of a pivot (the larger a pivot the better it is from a numerical point of view) and the fill in of the matrix (the less fill-in the more efficient future matrix manipulations will be). Default value is 1.0e-3. You are invited to change this value whenever Eldo informs that the matrix is singular.

- **PIVTOL**

  Is the absolute minimum value that can be accepted in the matrix to be a pivot for the LU-Factorization algorithm. Default is 1.0e-16. You are invited to change this value whenever Eldo informs that the matrix is singular.

- **PSOSC**=VAL

  Allows control of the maximum number of oscillations that can occur when using a DC pseudo-transient algorithm during DC. This is to avoid a never-ending loop in cases where the amplitude of the oscillations does not decrease. Default value is 10. If the user knows that the oscillations amplitude decreases after n periods, **psosc**=n can be specified to increase the Eldo limit.

- **QTRUNC**

  Forces Eldo to use a *Local Truncation Error* algorithm as in a SPICE like simulator. By default, the Eldo timestep is calculated from voltages using a predictor-corrector algorithm. This algorithm is suitable for IC circuits, however, for PCB-like circuits, it is preferable to use a *Local Truncation Error* timestep algorithm calculated from charges (and flux) as in SPICE like simulators. **QTRUNC** is automatically switched on if the circuit contains large current sources of above 1A, magnetic models or large power supplies. The option **NOQTRUNC** may be used to disable this algorithm. The *Local Truncation Error* algorithm, when working on charges or fluxes, uses the same control variables as the **GEAR** option, (that is, **RELTRUNC**, **CGHTOL**, **FLUXTOL** and **NGTOL**).

- **RATPRINT**=VAL

  If specified, then **DELMAX** is computed as:

  $$min\left(\frac{TSTOP}{50}, RATPRINT \times TPRINT\right)$$

  If not specified, then the default algorithm is used and **DELMAX** is not computed from **.TRAN** specifications.

- **RELTOL**=VAL

  Controls both the timestep size and the accuracy of Newton and/or IEM iterations. For voltages, convergence of iterations is reached when:

  $$|V(i) - V(i-1)| < RELTOL \times |max(|V(i)|, |V(i-1)|)| + VNTOL$$

  where ***V***(***i***) is the voltage value at current iteration ***i*** and ***V***(***i***–1) is the previous iteration. For currents, circuit convergence is reached when:

  $$|I(i) - I(i-1)| < RELTOL \times max(|I(i)|, |I(i-1)|) + VNTOL \cdot ITOL$$

  where ***I***(***i***) is the current value at voltage iteration ***i*** and ***I***(***i***–1) is the previous iteration. Default value is 1.0×10^{-3}.

  RELTOL also controls the time step size via the Local Truncation Error (LTE). This feature was added for compatibility with SPICE.

- **RELTRUNC**=VAL

  Is the relative tolerance on the **CHGTOL**, **NGTOL** and **FLUXTOL** parameters above. Default value is set by **EPS**.

- **RELVAR**

  Used for timestep control. Sets the relative voltage change for the condition that **LVLTIM**=1 and **DVDT**=0. If the nodal voltage at the present time point exceeds the nodal voltage at the last time point by **RELVAR**, the next timestep is reduced and a new solution is calculated. Default is 0.15.

- **SAMPLE**=tval

  May be used to speed-up simulation of sampled circuits containing **OPA** macromodels and/or switch macromodels. The simulation timestep is set to the value tval specified. Eldo then computes additional intermediate timesteps (that is, at tval/2, 3tval/2, 5tval/2, ...), in order to compute values when the signals should have reached their steady state values. The **SAMPLE** option also causes simplified **OPA** and switch macromodels to be used. The slew-rate of the simplified **OPA** macromodel is assumed to be infinite, and the simplified switch resistor value is either RON or ROFF, with no linear transition in-between.

- **SPLITC**[=val]

  Simulation of floating capacitors (for example, those derived from extraction) can be optimized by changing the value of this option. This will force Eldo to split a capacitor object into two capacitors, each connected between ground and one node of the original capacitor. This leads to faster, but less accurate simulations. This splitting occurs for capacitors with values greater than *val*. Default is 50fF (1nF if **DIGITAL** option is specified).

  This option is also used with ADiT in a slightly different way. A floating capacitor affects the partitioning between Eldo and ADiT solvers, if it connects two blocks that are assigned to the two different engines. Using this option allows the defined partitioning if capacitor value is less than *val*. In ADiT, *val* defaults to 1fF.

  When **SPLITC** is specified with a value, the same value is used as a threshold for both the capacitor splitting in Eldo and the ADiT partitioning.

  When **SPLITC** is specified without a value, then all the default values are used (including for ADiT partitioning).

  When **SPLITC** is not specified at all, then no splitting of capacitors occurs in Eldo (floating capacitors remain floating), however the default 1fF value is used as a threshold for ADiT partitioning.

- **STARTSMP**=VAL

  Used in conjunction with **FREQSMP**. The **FREQSMP** command will be active after **STARTSMP**.

- **STEP**=VAL

  Imposes a fixed timestep to be used by Eldo as defined by the VAL value. By default, Eldo uses a varying timestep.

Note

.OPTION STEP could produce incorrect results near breakpoints when used with the IEM method. This can occur if the **STEP** value is set greater than the rise and fall times of the circuit.

- **TIMESMP**=VAL|{VAL1,VAL2,...VALn}

  Forces Eldo to compute a time point at every multiple time interval of **TIMESMP**, the sampling time interval. Equivalent to 1.0/**FREQSMP**. Useful when performing a Fourier analysis. Multiple values can be specified as a list in which case Eldo computes timepoints corresponding to all sampled points. For example: a design has two clocks at 2MHz and 320 kHz. For uniform sampling, time sampling of 0.5µs and 3.125µs are needed to calculate the necessary exact points (for FFT post processing). Note that if options which impose the sampling frequency are set (for example interpolate=0 on **.OPTFOUR**), the timesmp array is reset to a single value.

- **TRTOL**=VAL

  Used for timestep control. Serves as a multiplier of the internal timestep generated by the Local Truncation Error timestep algorithm (**LVLTIM**=2). It is a factor that estimates the amount of error introduced in truncating a series used in the algorithm. This error is a reflection of what the minimum value of the timestep should be to reduce simulation time and maintain accuracy. The larger **TRTOL** is, the larger the timestep will be. Default value is 7.0.

- **TUNING**=[**FAST**|**STANDARD**|**ACCURATE**|**VHIGH**]
  [**TUNING**=**BACKANNOTATE**]

  Selects the default mode of operation with regards to precision and speed. This **TUNING** option acts as a macro-controller of Eldo: it enables the selection of algorithm and parameter settings via **EPS**. Any given parameter can be explicitly imposed, which will supersede the value related to the tuning setting. Note: the order of the option is important, the tuning type must be set first and then you can adjust any parameter with specific values. This option is equivalent to the `-tuning` command-line flag of Eldo. The `-tuning` flag overrides any **TUNING** option settings inside the netlist.

  FAST keeps Newton as the default algorithm, unless options **OSR** or **IEM** are explicitly imposed. **FAST** gives milder values to some tolerance parameters (**ITOL**, **VNTOL** and **ABSTOL**, but not **EPS**) in order to have faster but still reliable simulation, even if slightly less accurate. Well suited to efficiently simulate large analog or mixed circuits.

  STANDARD (this is the default) causes **EPS** to be set to 5.0e-3 and activates the **NEWTON** option unless options **OSR** or **IEM** are explicitly imposed.

  ACCURATE causes **EPS** to be set to 1.0e-6 and activates the **NEWTON** option unless options **OSR** or **IEM** are explicitly imposed.

  VHIGH causes **EPS** to be set to 1.0e-8 and activates the **NEWTON** option unless options **OSR** or **IEM** are explicitly imposed.

BACKANNOTATE enables an improved Eldo solver to be used for handling backannotated netlists with many parasitic elements. Backannotate can provide significant capacity and speed improvements (up to 10×) for DC, TRAN and all RF analyses for circuits which contain parasitic elements. It is most efficient when the parasitics are defined in DSPF format in a separate file. This can be specified in addition to one of the other keywords above. When specified alone, Eldo will automatically set **TUNING**=**STANDARD**. The classical tuning values are for Eldo accuracy and speed, and backannotate is for speed with parasitics usage. The **TUNING** option only accepts one argument at a time, so the option has to be repeated when combining arguments, for example:

```
TUNING=ACCURATE TUNING=BACKANNOTATE
```

For AC and transient noise (**.NOISETRAN**) analyses the backannotate algorithm is deactivated due to some inaccuracies in results. It can be forced by specifying **TUNING**=**AC_BACKANNOTATE**, however results cannot be guaranteed.

For greater flexibility, the backannotate algorithm can be activated or deactivated for a particular analysis by using one or several of the following options:

TUNING=**DC_BACKANNOTATE** (activates the solver for DC analysis)
TUNING=**AC_BACKANNOTATE** (activates the solver for AC analysis)
TUNING=**TRAN_BACKANNOTATE** (activates the solver for Transient analysis)
TUNING=**SST_BACKANNOTATE** (activates the solver for all RF analyses)
TUNING=**NODC_BACKANNOTATE** (deactivates the solver for DC analysis)
TUNING=**NOAC_BACKANNOTATE** (deactivates the solver for AC analysis)
TUNING=**NOTRAN_BACKANNOTATE** (deactivates the solver for Transient analysis)
TUNING=**NOSST_BACKANNOTATE** (deactivates the solver for all analyses)

Note

TUNING can also be specified using the time window .OPTPWL or .OPTWIND commands to apply or disable it during or outside some time intervals. The **BACKANNOTATE** tuning options do not support these commands.

- **UNBOUND**

  Enables an **EPS** value smaller than 1.0×10^{-10} to be specified. This option may be used for special applications that use low currents. Care must be taken, however, as problems with convergence may occur due to accumulation of round-off errors.

- **VMIN**=x1,
  VMAX=x2

  Sets the minimum and maximum voltage values for which Eldo searches for the DC operating point of a circuit. Power supply levels are very often parametrized, and the values of **VMIN** and **VMAX** typically depend on the power supply. Therefore **VMIN** and **VMAX** can have their values specified by parameters. For example:

```
.param foo = 1.0
.option vmin = '-10*foo' vmax = '10.0*foo'
```

- **VNTOL**=VAL

  Controls the voltage accuracy of the simulator when solving circuits using Newton Raphson techniques. Circuit convergence is reached when:

  $$(|V(i) - V(i-1)| < RELTOL \times \ |max(|V(i)|, |V(i-1)|)| + VNTOL)$$

  where ***V***(***i***) is the voltage value at current iteration ***i*** and ***V***(***i***–1) is the previous iteration. Default value is 1µV. Note **ABSV** is synonymous to **VNTOL**.

- **WDB_IDELTA**=VAL

  This option is used to define the delta I for use with all current plots, using this option will reduce the size of the wave database. Only for use with the JWDB (*.wdb*) output format. Default value is 0.0.

- **WDB_VDELTA**=VAL

  This option is used to define the delta V for use with all voltage plots, using this option will reduce the size of the wave database. Only for use with the JWDB (*.wdb*) output format. Default value is 0.0.

- **WDB_NOSYNCHRO**=0|1

  When set to 1 this option forces Eldo to send unsynchronized waves to the database using default VDELTA and IDELTA values. This will reduce the size of the wave database. Only for use with the JWDB (*.wdb*) output format. Default value is 0.

- **XA**=VAL

  Diffusion length for MOS S/D calculation. Default value is 6µm.

  $$Weff = W - DW - 2 \times kl$$
  $$ADeff = Weff \times xa$$
  $$ASeff = Weff \times xa$$
  $$PDeff = Weff + 2 \times xa$$
  $$PSeff = Weff + 2 \times xa$$

Miscellaneous Simulation Control Options

- **AMMETER**

  Prevents Eldo from eliminating zero voltage sources which are frequently used as current probes.

- **AUTOSTOP**=0|1|2

  This option can be specified to reduce the simulation time. It stops a Transient simulation, when the Transient Extraction Language functions **TPD**, **TRISE**, **TFALL**, **TCROSS** are used. It can also be used with the General Extraction Language functions: **XDOWN**, **XUP**, **YVAL**, **XTHRES**. For any other functions present in the netlist, the **AUTOSTOP** specification will be ignored.

AUTOSTOP=0
Deactivates autostop if necessary. Default.

AUTOSTOP=1
Causes Eldo to stop the simulation when all extracted waveform information (**.EXTRACT**/**.MEAS**) has been measured.

AUTOSTOP=2
Used in multi-step simulations. Causes Eldo to stop when all sweep measurements are completed.

- **AUTOSTOPMODULO**=VAL

  This option is for use with the **AUTOSTOP** option. It only has an effect in TRANSIENT analysis. The evaluation of MEAS/EXTRACT will be performed only at every **AUTOSTOPMODULO** steps. This can be used to overcome the simulation slow down sometimes caused by setting **AUTOSTOP**, especially when the time spent to evaluate the MEAS/EXTRACT is large compared to the time taken to solve for the circuit. Default is 0, meaning that evaluation is performed at each step.

- **CARLO_GAUSS**

  Option for Gaussian distribution in Monte Carlo analysis. This option has two effects:

  - All **LOT**/**DEV** statements having no distribution type specification, will be assumed to have a Gaussian distribution, that is, default is:

    LOT/**GAUSS** or **DEV**/**GAUSS**

    Example:

    ```
    VT0 = 1 LOT/UNIFORM = 20%
    EOX = 10n LOT = 12%
    .OPTION CARLO_GAUSS
    ```

    In this example the distribution on **EOX** will be Gaussian, while the distribution on **VTO** will remain uniform.

  - The envelope returned in the binary output files (or in the **.PRINT** commands) is computed from the standard deviation and do not correspond anymore to (**MIN**, **MAX**) of the waveform.

- **CPTIME**=VAL

  Stops the simulation if the CPU time exceeds VAL seconds. Message is displayed in the *.chi* file as well as being sent to the screen.

- **DEFAULTFALLTIME**=VAL

  Transition time in seconds for signal changing from high to low state. Used as a default by test vector files, see ".TVINCLUDE" on page 918. Default value is 1e-10.

- **DEFAULTRISETIME**=VAL

  Transition time in seconds for signal changing from low to high state. Used as a default by test vector files, see ".TVINCLUDE" on page 918. Default value is 1e-10.

- **DEFPTNOM**

  Allows a parameter to be defined with the name TNOM. In such a case, this value will be used inside parameter expressions instead of the default **TNOM** or the value set using option **TNOM**=val. This option must be specified at the top of the netlist. The temperature value used by the Eldo model evaluator is always that set with option **TNOM**=val.

  In the following example, the voltage source will use the value specified by tnom in the **.PARAM** command; without option **DEFPTNOM** the default **TNOM** value would be used:

```
.option defptnom
...
.param tnom=50
v1 1 0 tnom
```

- **DSCGLOB=X|GLOBAL**

  X
  This is the default. Nodes which appear in subcircuit definitions have priority over the nodes defined with **.GLOBAL** statements. See ".GLOBAL" on page 676. This option must be specified at the very beginning of the netlist.

  GLOBAL
  When specified, nodes defined with **.GLOBAL** statements have priority over the nodes which appear in subcircuit definitions. See ".GLOBAL" on page 676. This option must be specified at the very beginning of the netlist.

  See "DSCGLOB Option Example" on page 903.

Note

When Eldo finds a subcircuit definition which uses global nodes, Eldo issues a warning to inform you how the situation is handled.

- **DSPF_LEVEL=C|RC|RCC**

  Defines the level of parasitics to use from a specified DSPF file. The option will use part of the information stored in the RC extracted DSPF file specified by **C**, **RC** or **RCC**.

  DSPF_LEVEL=C
  Where **C** is the intrinsic and coupling capacitance. The pre-layout nets will obtain a grounded capacitance, as specified in the DSPF file in the line ***|NET**.

  DSPF_LEVEL=RC
  Where **RC** is the distributed RC elements. The ideal pre-layout nodes will be replaced with RC elements. Also the coupling capacitance between wires/layers is ignored, that is any capacitor that is not grounded (floating) will be ignored.

  DSPF_LEVEL=RCC
  Where **RCC** is both intrinsic and coupling capacitance and distributed RC elements.

  It is assumed that the DSPF file has been extracted with all RCC information.

Note

This option can also be specified by the **LEVEL** parameter in the **.DSPF_INCLUDE** command, see “.DSPF_INCLUDE” on page 617. If specified in the command it will overwrite this option.

- **FLOATGATECHECK**

  If a node is connected to at least one MOS gate, and is not connected to anything else but MOS gates, capacitors, or a reverse-biased diode, it is considered as a floating gate.

  This option enables Eldo to issue a warning when a floating gate is detected and resume the simulation. Eldo will not consider reverse-biased diodes as active elements when checking for floating gates. Reverse-bias conditions are detected as follows:

  - If the node is connected to the first pin of a diode, then the other pin must be connected to the positive node of a power supply to be considered as reverse-bias.
  - If the node is connected to the second pin of a diode, then the other pin must be connected to either ground or the negative pin of a power supply to be considered as reverse-bias.

  This option can be used in conjunction with the options **FLOATGATERR** and **FLOATGATE0**.

- **FLOATGATERR**

  Enables Eldo to stop the current process and generate an error when floating gates are detected. It can be used in conjunction with the option **FLOATGATECHECK**.

- **FLOATGATE0**

  This option will force detected floating gates to 0. It can be useful to change the topology of a circuit and achieve better convergence. It can be used in conjunction with the option **FLOATGATECHECK**.

- **FALL_TIME**=VAL

  Transition time in seconds for signal changing from high to low state. Used by the interactive LOW command. Default value is 1ns.

- **RISE_TIME**=VAL

  Transition time in seconds for signal changing from low to high state. Used by the interactive HIGH command. Default value is 1ns.

- **HIGHVOLTAGE**=VAL

  Sets the upper bus signal voltage level. Default is 5V. This option (and **LOWVOLTAGE**) is only used by the **.TVINCLUDE** command and by interactive commands.

- **LOWVOLTAGE**=VAL

  Sets the lower bus signal voltage level. Default is 0V. This option (and **HIGHVOLTAGE**) is only used by the **.TVINCLUDE** command and by interactive commands.

- **LOWVTH**=VTH1

  See below. Default value is 2.4V. This option (and **HIGHVTH**) is only used by the **.TVINCLUDE** command.

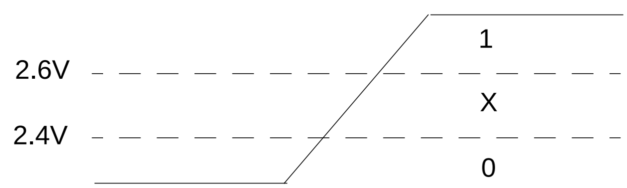
- **HIGHVTH**=VTH2

  Default value is 2.6V. This option (and **LOWVTH**) is only used by the **.TVINCLUDE** command.

  The VTH parameters are required by Eldo to compute the **HEX** (or **DEC**, **OCT**, or **BIN**) value on the bus from the analog value inside Eldo. Then, it compares this value with the value expected and displays the error if they are not the same.

  When only vth1 is given:
  If value < vth1 then logic state 0.
  If value > vth1 then logic state 1.

  HighVth=vth2 is used to plot the indeterminate value as shown below:
  If value < vth1 then logic state 0.
  If vth1 < value < vth2 then state X.
  If value > vth2 then logic state 1.



- **INTERP**

  Causes Eldo to generate data in the binary waveform output file in the same way it does in the *.chi* file. Therefore, rather than producing data as it is calculated it will only produce points at each timestep specified in the analysis. Example:

```
.TRAN 1n 10n
.OPTION INTERP
```

  This will generate data in the binary output file every 1ns. If **INTERP** is not specified, then Eldo will dump points that it has actually computed.

- **ICDC** and **ICDEV**

  By default, Eldo behaves like Spice 2g6 regarding **.IC** commands and **IC** parameters specified on devices: **.IC** commands are taken into account only for DC done before Transient analysis, or when **.TRAN ... UIC** is specified. However, **IC** parameter specifications on devices are taken into account only in the case of **.TRAN ... UIC**.

- **ICDC**

  In this case, **.IC** commands (and **IC** parameter specifications on devices when option **ICDEV** specified) will be taken into account for any DC analysis.

- **ICDEV**

  In this case, **IC** parameters specified on devices and **.IC** commands will be handled the same way.

- **LICN**

  By default, the first initial condition (**.IC**) specification has precedence over subsequent IC specifications. Setting **LICN**, the last IC specification will have precedence. This option is automatically set if compat mode is used.

  The same remark applies to **.NODESET**, or **.GUESS**. Setting **LICN**, the last **.NODESET** (or **.GUESS**) specification will have precedence over the previous **.NODESET** (or **.GUESS**) specifications. See ".IC" on page 682, ".GUESS" on page 677, and ".NODESET" on page 743.

Note

Whether or not **LICN** is specified, **.NODESET** always has precedence over **.GUESS**, and whenever **.IC** commands are active (that is, during the DC which is done prior to TRANSIENT simulation, or whenever option **ICDC** is active), then **.IC** has precedence over **.NODESET**.

- **LVS_IGNORE_VARIABLE**="names"

  Specifies a list of LVS (Layout Versus Schematic) variables that Eldo will ignore for MOS, BJT, diode, and JFET devices. Specify the variables in a list separated by commas. For example:

```
.OPTION IGNORE_LVS_VARIABLE = X, Y, D
```

  Eldo ignores the $X, $Y, and $D LVS variables and associated values.

- **M53**

  Specifies the ruling surrounding the M factor should be as it was for versions of Eldo before and including v5.3. Example:

```
.SUBCKT mysubckt in out m=5
r1 in 1 1k m='2*m'
r2 1 out 1k
.ends
X1 in out mysubckt m=3
```

  The above example used to work for previous versions of Eldo (Eldo would simulate as if there were six X1.r1 devices in parallel, but only one X1.r2.). For Eldo version v5.4 and higher, the rule regarding the **m** factor changed. The **m** value can appear on the X instance or on the device. The **m** value cannot appear anymore on the Right-Hand-Side of the expression. In the example above, Eldo will issue an error. To avoid this error, it is therefore mandatory to replace the **m** parameter in any expression, by another parameter name. In the example below, no confusion is possible:

```
.SUBCKT mysubckt in out mr=5
r1 in 1 1k m='2*mr'
r2 1 out 1k
.ends
X1 in out mysubckt mr=3
```

Here, the user will still have six devices of `X1.r1`, and only one of `X1.r2`. Of course, if the user put the **m** factor on the subcircuit instantiation:

```
X1 in out mysubckt m=3 mr=3
```

Here, 18 devices of `X1.r1`, and three devices of `X1.r2` will be emulated.

For backward compatibility, you can invoke Eldo with the `-m53` flag, or use option **m53** (to be placed at the beginning of the *.cir* file, just after the title).

- **MC_IGNORE_BINNING**

  Disables automatic selection of the binning parameters at run time; the model selected at the nominal run will then be used for the whole MC process. See Define Monte Carlo Parameters.

- **MMSMOOTH**

  Use this option to make **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** operators derivable, to avoid convergence problems. The **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** operators, when used in bias-dependent expression (such as in R(V)), can cause non-convergence due to these operators making the function non-derivable.

  With option **MMSMOOTH**:

$$MIN(a,b) = a - 0.5 \times \left((a-b) + \sqrt{(a-b)^2 + eps}\right)$$

$$MAX(a,b) = b + 0.5 \times \left((a-b) + \sqrt{(a-b)^2 + eps}\right)$$

$$ABS(val) = \sqrt{val^2 + eps}$$

$$SIGN(val) = -1 \qquad \text{when } val < -eps$$

$$SIGN(val) = \sin\left(\frac{\pi}{2} \times \frac{val}{eps}\right) \qquad \text{when } eps < val < eps$$

$$SIGN(val) = 1 \qquad \text{when } val > eps$$

  eps is the smoothing coefficient, which can be set with option **MMSMOOTHEPS**=`val`. Default is 1.0e-3.

  Notes:

  - setting **MMSMOOTHEPS**=`val` automatically activates **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** operator smoothing, that is, it emulates the specification of option **MMSMOOTH**

- **MMSMOOTH** works only on bias-dependant expressions used on devices, it has no effect on other types of expressions (**.DEFWAVE** for example)
- Smooth **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** operators also exist as built-in operators: **SMMIN**(a,b,eps), **SMMAX**(a,b,eps), **SMABS**(val,eps), **SMSGN**(val,eps), and **SMSIGN**(val,eps); see "Arithmetic Functions and Operators" on page 78

- **MMSMOOTHEPS**

  Specifies the smoothing coefficient, *eps*, used to make **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** operators derivable. Default is 1.0e-3. Setting **MMSMOOTHEPS**=val automatically activates **MIN**/**MAX**/**ABS**/**SGN**/**SIGN** operator smoothing, that is, it emulates the specification of option **MMSMOOTH**. See also option MMSMOOTH above.

- **NOICNODE**

  Causes Eldo to ignore any **.IC** commands.

- **NOLICN**

  Causes the first initial condition (**.IC**) specification to have precedence over subsequent IC specifications. If running Eldo in compat mode, option **LICN** is automatically set causing the last initial condition (**.IC**) specification to have precedence over previous IC specifications. Use option **NOLICN** to disable this behavior for backward compatibility (pre-v6.9).

- **NOLTEDISC**

  Bypass LTE checks on those devices where convergence problems or slow simulation are observed. When using operators such as CEIL, ROUND, FLOOR on bias dependant expressions, for example E 1 2 value = {round(v(in) + 2}, convergence problems may be caused because the functions are not continuous. When Eldo encounters such objects, it will generate a warning about these possible convergence problems, and will invite the user to use this option.

- **NOMEMSTP**

  Setting this option means Eldo will not use the results of the previous STEP run as an initial guess for the next one. By default, without this option, the second run in **.STEP** uses the result of the previous STEP as an initial guess.

- **NOVATOPOCHK**

  Bypass topology checks for Verilog-A module instances. Beware that voltage loop and DC path to ground dection procedures for the whole circuit may be affected by this option if there are Verilog-A instances.

- **PARAMOPT_NOINITIAL**

  This option can be used for the Eldo optimizer. Allows the initial value parameter of the **.PARAMOPT** syntax to be omitted. The value already available in the netlist through the normal **.PARAM** specification will instead be used.

- **PODEV**

  This option is used to specify **DEV** variation for a declared Monte Carlo analysis distribution parameter when the parameter is used as part of either a model or instance parameter. By default **LOT** variation is used for model parameters and **DEV** is used for instance parameters. In Eldo versions prior to v6.3_2 **DEV** variation was the default for both model and instance parameters.

- **RANDMC**

  Reset the MC randomly. The sequence of random numbers differs from one run to the next.

- **RGND**=`val`

  When this option is set a resistance is connected between a node and ground for nodes that have no DC path to ground (warning message 118), for example a node to which only capacitors are connected. This allows the simulation to proceed instead of producing errors. It differs from option RGNDI which is used only for dangling nodes connected to a current source. This option connects a resistance of `val` and allows the simulation to proceed.

- **RGNDI**=`val`

  When this option is set a resistance is connected between the dangling nodes of independent current sources and ground. Normally if Eldo detects a dangling node on a current source an error is issued and the simulation stops. This option connects a resistance of `val` and allows the simulation to proceed.

- **SIGTAIL**=`VAL`

  This option can be used in Monte Carlo analysis. The value represents the relative maximum extension of the Gaussian tail when Gaussian distribution is used. The default is 4. Example:

  ```
  .param p1=1 lot/gauss=0.1
  .option sigtail=6
  ```

  Here, `p1` values will be in the range [0.4, 1.6] (which is equivalent to nominal-0.6, nominal+0.6). If **SIGTAIL** is not specified, `p1` values will be in the range [0.6, 1.4] (equivalent to nominal-0.4, nominal+0.4).

- **STATISTICAL**=`0|1`

  Specifies whether any statistical variation due to **.MC**, **.WCASE**, or **.DCMISMATCH** can be applied to X instances, device declarations, or **.SUBCKT** definitions. If **STATISTICAL** is 0, the selected devices will keep their nominal values. If **STATISTICAL** is 1, the selected devices have statistical variation applied. Default is 1.

- **TEMP_UNIT**=**KELVIN|CELSIUS**

  Specifies the unit used for temperature values inside the netlist. Default is Celsius. This has an effect on commands **.STEP TEMP** `<t1> <t2>` and **.TEMP** `<t1>` and on the unit used to print temperature values in the outputs: simulation information, **.EXTRACT** results, in *.aex* file, and so on.

- **TNOM**=VAL

  Sets the model temperature, that is, the temperature defined in the model. Default is 27°C.

- **TPIEEE**

  For each port for which **NOISETEMP** is not specified, then the value 16.85 will be used rather than the temperature of the circuit. See "NOISETEMP" on page 315 and "NOISETEMP" on page 315 in the V&I Sources.

- **ULOGIC**

  Causes a current source/RC combination to be used on the output of Eldo logical primitives **AND**, **NAND**, and so on. By default, the output of Eldo logic gates acts as a pure voltage source, making simulation very fast, but not allowing easy arbitration of signals when logical outputs are connected together. By using **ULOGIC**, depending on the values of driving resistance used, such conflicts may be resolved. Note that in this mode of operation, simulation speed is slightly slower than the default voltage source method.

  See "Digital Model Definition" on page 449 for details of output resistor and propagation delay model parameters.

- **ZOOMTIME**=VAL

  The hold times of **PWL**, **PULSE** and **PATTERN** sources are extended by multiplying them by a factor VAL. Source rise and fall times remain unchanged.

Model Control Options

- **ACDERFUNC**

  Derivative of function selector. For AC analysis only. For voltage or current dependent R/L/C devices, the derivatives of the value with respect to the voltage are dumped in the matrix by default. By default this option is set (prior to Eldo v6.8 it was not). This can be disabled by globally setting **NOACDERFUNC** (see "NOACDERFUNC" on page 989), forcing Eldo to not dump these values in the matrix. Device objects can be individually set with the **ACDERFUNC** local parameter. The specification on the device overrides any global setting by options **ACDERFUNC** or **NOACDERFUNC**.

- **ACM**

  Usually **ACM** is just a parameter which is an alias to **ARLEV**; that is, it overwrites the values of **ALEV** and **RLEV**. However, if you specify **ACM** in the netlist, then **ACM** has a much more effective role. This option is automatically set if compat mode is used.

  Please refer to .OPTION ACM of the *Eldo Device Equations Manual*.

- **ASPEC**

  Controls use of **SCALE** and **SCALM** values with the models. When this option is specified, the following conditions are set:

  ```
  .OPTION SCALE = 1U
  .OPTION SCALM = 1U
  .OPTION WL
  ```

This option is intended for use with ASPEC MOSFET models (Eldo level 16), but also impacts other models because the options set are global.

- **BSIM3VER**=VAL

  There are several versions of the BSIM3v3 model. See "Berkeley SPICE BSIM3v3 Model (Eldo Level 53)" on page 273. This option is an alternative to the model parameter **VER** for setting the version to use. By default, BSIM3v3.2.4 (**VER**=3.24) is selected.

- **DEFAD**=VAL

  Sets the default value for MOS drain diffusion area. Default is zero.

- **DEFAS**=VAL

  Sets the default value for MOS source diffusion area. Default is zero.

- **DEFL**=VAL

  Sets the default value for MOS channel length. Default is 100µm.

- **DEFW**=VAL

  Sets the default value for MOS channel width. Default is 100µm.

- **DEFNRD**=VAL

  Sets the default value of MOS parameter **NRD**. Default is zero.

- **DEFNRS**=VAL

  Sets the default value of MOS parameter **NRS**. Default is zero.

- **DEFPD**=VAL

  Sets the default value for MOS drain perimeter. Default is zero.

- **DEFPS**=VAL

  Sets the default value for MOS source perimeter. Default is zero.

- **ELDOMOS**

  Sets option **ACM** together with the items listed below:

  - sets the default value of ***TNOM***=25 instead of the normal 27
  - if ***LD*** is not specified, $LD = 0.75 \cdot XJ$
  - $Ids = Ids + gmin \cdot Vds$
  - for ***RLEV***=5: (if ***defnrd*** & ***defnrs*** not defined)
    sets the default value of ***NRD***=0
    sets the default value of ***NRS***=0
  - if (accusim2), sets a lower limit for ***Isat*** which is 1×10^{-18} for BSIM2 and 1×10^{-15} for any other model
  - calls the impact ionization model if !accusim2 & !BSIM2

- o required for the usage of LEVEL 17 (SPICE-A LEVEL 8)
- o allows the separate use of both ***COX*** and ***TOX*** at the same time for Levels 1, 2 & 3
- o for Level 2:
  uses ***CAPLEV***=4 for the calculation of the intrinsic charges & capacitances
- o for Levels 1, 2, 3, 16 & 17:
  $Leff\ =\ Ldrwan \cdot LMLT + XL - 2(LD + DEL)$
  $Weff\ =\ Wdrwan \cdot WMLT + XW - 2WD$
- o for Levels 1, 2, 3, 16 & 17:
  uses SPICE-A mobility temperature equations
  uses SPICE-A surface potential (***Phi***) temperature equations according to ***TLEVC***
  uses SPICE-A $VBI \cdot VTO$ temperature equations according to ***TLEV***
- o for the AMS model Level 15:
  uses another capacitance model rather than SPICE 2G6 if ***CAPOP***=0
- o switches automatically:
  Level1 $\Rightarrow$ Level 11
  Level2 $\Rightarrow$ Level 12
  Level3 $\Rightarrow$ Level 13
- o if !BSIM2 & !MOSP9: adds 2***WD*** to ***Weff*** used for the calculation of the overlap capacitance to cancel the effect of the already subtracted 2***WD***.

- **FNLEV**

  Selects between the two flicker noise models of the BSIM3v3 model with **NOIMOD**=1&4.

- **GMIN**=VAL

  Sets the conductance value that is placed in parallel with all PN junctions and drain and source nodes of MOSFET models. It enhances the convergence properties that are degraded by having too low a value of OFF conductance for PN junctions and MOSFET devices. Large values of **GMIN** may cause unreasonable circuit response. Default is 1.0×10^{-12}.

- **GMIN_BJT_SPICE**

  A defect was fixed in the BJT level 1 and 5 models for the Gmin handling. This defect caused unrealistic high leakage current values. This defect was also found in the original Spice code. The fix affects the backward compatibility of results in some test cases. Use option **GMIN_BJT_SPICE** to reproduce results for backward compatibility.

- **GMINDC**=VAL

  Similar to option **GMIN**, however the value specified via **GMINDC** is used for DC analysis only. Defaults to **GMIN**.

- **GRAMP**=VAL

  Ramps the conductance value that is placed in parallel with all PN junctions of all devices, and all drain and source nodes of MOSFET models, from **GMIN**$\times10^{\mathrm{GRAMP}}$ down to **GMIN**.

This helps DC convergence in some circuits. This option may also be used in conjunction with the **.RAMP** command. Default value is 0. See ".RAMP" on page 852.

- **GENK**[=VAL]

  Forces Eldo to generate 2nd order mutual inductors. It is activated if `val` is any value greater than zero or if no value is stated. If option **COMPAT** is set it is automatically activated and can be deactivated using **GENK**=0. Used together with the **KLIM** option.

- **HIER_SCALE**=0|1

  "S" parameter subcircuit scaling. Forces Eldo to overwrite the global SCALE by the parameter "S" specified in an X instance call. By default, option SCALE is global and applies to all instances. For example:

```
.option SCALE=1u HIER_SCALE=1
.subckt foo d g
m1 d g 0 0 n w=100 l=50
.ends

.subckt foos d g
m1 d g 0 0 n w=10 l=5
.ends
x1 1 2 foo
x2 1 2 foos s=10u
```

  The width, `w`, of X1.M1 will be computed as 100×SCALE = 100u. The width, `w`, of X2.M1 will be computed as 10×X2×s = 10×10u = 100u.

- **IBIS_SEARCH_PATH**="FILEPATH"

  Specifies the path to the directory to search for IBIS files.

- **KLIM**=VAL

  Used together with the **GENK** option to force Eldo to generate 2nd order mutual inductors. Default is 0.02.

  In the case of the following example:

```
K1 L1 L2 val1
K2 L1 L3 val2
```

  by specifying the **GENK** option, a new mutual `K3` between `L2` and `L3` will be generated, if there is no such mutual already present in the netlist. The value of the new mutual created would be `val1`×`val2`, and the mutual coefficient is only created if `val1`×`val2` is higher than the value of **KLIM**. However, the new mutual `K3` will not be generated if:

  - **GENK** is equal to zero,
  - the mutual coefficient is lower than the value of **KLIM**, or
  - a mutual is already present in the netlist.

  A list of the mutuals, which are automatically created, is dumped in the ASCII output file.

- **MAXADS**=VAL

  Sets the maximum value for the MOS source diffusion area or drain diffusion area. No default is specified.

- **MAXL**=VAL

  Sets the maximum value for the MOS channel length. No default is specified.

- **MAXPDS**=VAL

  Sets the maximum value for the MOS source diffusion perimeter or drain diffusion perimeter. No default is specified.

- **MAXW**=VAL

  Sets the maximum value for the MOS channel width. No default is specified.

- **MINADS**=VAL

  Sets the minimum value for the MOS source diffusion area or drain diffusion area. No default is specified.

- **MINL**=VAL

  Sets the minimum value for the MOS channel length. No default is specified.

- **MINPDS**=VAL

  Sets the minimum value for the MOS source diffusion perimeter or drain diffusion perimeter. No default is specified.

- **MINW**=VAL

  Sets the minimum value for the MOS channel width. No default is specified.

- **MINRACC**=VAL

  If the resistor value is below **MINRACC**, Eldo will not create access resistor of devices MOS, BJT, Diode or JFET. Default is undefined, that is, access resistors are always created.

- **MINRESISTANCE**=VAL

  Using this option is equivalent to using the options **RMMINRVAL** and **MINRVAL**. Resistors with a value less than the specified VAL will be removed *before* partitioning occurs and a zero voltage source will be inserted between the two pins, and then one of the pins removed. The resistor cannot be reactivated when this option is used. Resistors that are connected to a pin which appears in a **.SUBCKT** line, will not be removed.

  To allow a resistor to override this option use the **KEEPRMIN** parameter, see “KEEPRMIN” on page 125. For resistors connected directly to a voltage source, see option **RAILRESISTANCE**.

- **MINRVAL**=VAL

  Removes resistors with absolute values below val. It emulates a zero voltage source between the two pins of the resistor. By default, **MINRVAL** is not specified. This means that no action will be taken regarding resistor devices.

To allow a resistor to override this option use the **KEEPRMIN** parameter, see "KEEPRMIN" on page 125.

Note

The connection is not allowed in a case where suppression of the resistor would create a Voltage loop. For the same reason, (that is, prevention of a Voltage loop) resistors connected to Y elements are never removed.
To prevent cases where the resistor value would become larger than **MINRVAL** in a new simulation, the resistor is reactivated. This could happen in instances where the resistor's value is controlled by a parameter, varying according to a **.STEP** command for instance.

- **MNUMER**

  Applies to MOS devices. Sets the derivative computation method to the finite difference method (DERIV=0). By default, DERIV=1 for analytical derivatives.

- **MOD4PINS**

  Sets the number of pins that must be specified in the MOS instantiation to 4. (The only model that can have more than 4 pins is the BSIMSOI3 model.) Setting this option has the effect that for the following line:

  ```
  M1 D G S B MOD W L AD AS PD PS NRD NRS M=<value>
  ```

  `MOD` is considered as the model name. Otherwise, Eldo would look for a model named `NRS`, with `M1` assumed to have 12 pins.

- **MODMONTE**[=0|1]

  Compliance with TSMC TMIv2. This option only has an effect on Monte Carlo analysis parameter variations specified via GAUSS/AGAUSS/UNIF/AUNIF. When set to 1, Eldo will extract a new random value for each usage of the parameter. When set to 0 (default), there is only one extraction per X instance.

- **MODWL**

  This option is not required because it is set by default. MOS model versions are selected via **.MODEL** command **W** and **L** parameters (binning parameters). Use option **NOMODWL** to switch off the functionality.

  For further details please see "Selection of MOSFET Models via W/L Specifications (Binning)" on page 257.

- **MODWLDOT**

  This option is used for binned models. By default, the extension of a binned model is either: `<root>.<extension>` or `<root>_<extension>`

  This can be confusing when `<root>` also contains the character "_". Specify option **MODWLDOT** to only allow "**.**" as the separator. When set, the character "_" can be used in the `<root>`. This option is automatically activated with the `-compat` flag. See also "Selection of MOSFET Models via W/L Specifications (Binning)" on page 257.

- **NGATEDEF**[=node_name]

  Specifies NMOS floating gates to be connected to the specified node name. Default is node 0.

- **NOACDERFUNC**

  The option **ACDERFUNC** is set by default (beginning Eldo v6.8). Specify **NOACDERFUNC** to disable this for backward compatibility. See also "ACDERFUNC" on page 983. Device objects can be individually set with the **ACDERFUNC** local parameter. The specification on the device overrides any global setting by options **ACDERFUNC** or **NOACDERFUNC**.

- **NOAUTOCTYPE**

  Disables automatic checking by Eldo of the dependencies of the capacitor value. Without this option, if Eldo finds that the capacitor value does not depend on the bias across the terminal of the capacitor, then Eldo will behave on that device as if CTYPE=1 had been set. With this option set, Eldo will behave as if CTYPE=0 (default) has been set, unless it is otherwise explicitly specified.

- **NOCATMX**

  Disables merging devices in parallel. This option is equivalent to the -nocatmx command-line flag. For further details see "Merging Devices in Parallel" on page 120.

- **NOMODWL**

  By default, MOS model versions are selected via **.MODEL** command **W** and **L** parameters (binning parameters). Specify this option to unset this behavior.

  For further details see "Selection of MOSFET Models via W/L Specifications (Binning)" on page 257.

- **NWRMOS**

  Forces all access resistors of MOS that are connected to nodes which are to be solved by NEWTON, to be created as objects, that is, as if the access resistors had been explicitly instantiated in the netlist. This is for improved accuracy. Default is **NWRMOS**.

- **PGATEDEF**[=node_name]

  Specifies PMOS floating gates to be connected to the specified node name. Default is node 0.

- **RAILINDUCTANCE**=VAL

  This option is similar to the **RAILRESISTANCE** option. Inductors connected directly to a power supply or ground and below the specified value VAL will be replaced by a short circuit. The idea is to remove effects of small inductance added between power supply and the actual circuitry in order to speed up simulation. The inductor cannot be reactivated when this option is used.

- **RAILRESISTANCE**=VAL

  This option is similar to the **MINRESISTANCE** option. Resistors connected directly to a voltage source and below the specified value VAL will be removed. A zero ohm resistor will be inserted between the two pins, and then one of the pins removed. The resistor cannot be reactivated when this option is used.

- **REDUCE**

  This option is set by default. Multiple identical resistors, diodes, BJTs, MOSFETs, and subcircuits that follow each other are reduced into a single instance using the device multiplier **M** parameter, for example:

```
X1 1 2 FOO A = 1 B = 1
X2 1 2 FOO A = 1 B = 1
X3 1 2 FOO A = 1.0 B = 1
```

  Here, X instances `X1` and `X2` will be replaced by:

```
X1 1 2 FOO A = 1 B = 1 M = 2
```

  but `X3` will remain as it is because the character string for A does not match.

  For more information see “Combining Identical Resistors” on page 126, “Combining Identical Diodes” on page 227, “Combining Identical BJTs” on page 234, “Combining Identical MOSFETs” on page 252, and “Combining Identical Subcircuits” on page 303 respectively.

- **RESNW**=val

  Resistors with values higher than **RESNW** are allowed to be solved by OSR. Default is 1.0e18 Ω (that is, virtually infinity).

- **RMMINRVAL**

  With this set option **MINRVAL**=val will emulate a **.CONNECT** through the 2 pins of the resistor. The advantage of this is that it reduces the number of nodes to be solved, but the disadvantage is that the resistor cannot be reactivated.

  To allow a resistor to override this option use the **KEEPRMIN** parameter. For more information see “KEEPRMIN” on page 125.

- **RMOS**

  When this is set, all MOS access resistors are unconditionally created as objects. Default is **NORMOS**.

Note

Whenever MOS access resistors are not created as objects, Eldo uses an iterative process to handle their effects. This process is less CPU time consuming than the resolution of the additional nodes which are created to connect the access resistors to the MOS devices.

- **RSMALL**=VAL

  Resistors with a value smaller than **val** will be set to **val**. Default is $1.0e^{-6}$ Ω.

- **RZ**=VAL

  This will set the *global* value (VAL) for **RZ** which will enable Eldo to detect a 'Z' state on *all* A2D nodes when the option ZDETECT is used. A 'Z' state is detected when the equivalent impedance of the A2D node exceeds the specified **RZ** value.

- **SCALE**=VAL

  Multiplier for MOS width, length, perimeter of drain and source. The parameters **AD** and **AS** are multiplied by x^2. The default value of **SCALE** is 1.0.

  For more information on these parameters, refer to the Device Models chapter. For more information on scale factors see "Scale Factors" on page 73. The keyword **SCALE** can be used in expressions. For more information on such keywords see "Reserved Keywords" on page 72.

  SCALE can also act as a multiplying factor for diodes, according to the value of the **SCALEV** parameter of the diode model, see Level 1 Scaling in the *Eldo Device Equations Manual*.

  For **.EXTRACT** function **EVAL**(device_name,device_entity) to take into account the option **SCALE** multiplier specify option **EXTRACT_EVAL_FINAL**. See "EXTRACT_EVAL_FINAL" on page 1000.

- **(NO)KWSCALE**

  SCALE can be considered as a keyword by Eldo (**KWSCALE** set), or not (**NOKWSCALE** is set). The default condition is as if option **KWSCALE** has been explicitly declared on the netlist, meaning that **SCALE** is a keyword. The default value of **SCALE** is 1.0.

  When **SCALE** is considered as a keyword, **SCALE** can appear in expressions, and its value will be that assigned via option **SCALE**=val.

  When **SCALE** is not considered as a keyword, **SCALE** can also appear in expressions, but its value must be defined via a **.PARAM** statement, as is the case for any other parameter.

  When -compat is set at invocation of the simulator, **NOKWSCALE** is assumed to be set, and can be switched with option **KWSCALE**.

  When -compat is not set at invocation of the simulator, **KWSCALE** is assumed to be set, and can be switched with option **NOKWSCALE**.

- **SCALEBSIM**=VAL

  Scales all sensitivity parameters of the BSIM1 and BSIM2 MOSFET models. For example, **VFB** is a basic parameter which is corrected by the length and width sensitivity parameters, **LVFB** and **WVFB**. **SCALEBSIM** will scale these parameters by the factor VAL. Default value is 1, for example:

$$vfb = VFB + (LVFB \cdot SCALEBSIM / L) + (WVFB \cdot SCALEBSIM / W)$$

- **SCALM**=VAL

  Scaling factor for the model parameters **LDIF**, **DL** and **DW** (MOS), **DW** and **DLR** (RC wire), for example **DL** is equivalent to the **LVAR** parameter for the MM9 models (see the "Philips MOS 9 Model (Eldo Level 59 or MOSP9)" on page 283). **SCALM** can be individually defined for each model card using the model parameter **SCALM**. This overrides the global **SCALM** value defined using the **.OPTION** command.

For more information on these parameters, refer to the Device Models chapter.

- **SHRINK_FACTOR**=VAL

  Specifies a value to multiply the **SCALE** value by to obtain the effective **SCALE** value. In case of multiple definitions, the effective **SCALE** value is the product of the last specified **SCALE** value and the last specified **SHRINK_FACTOR** value. **SHRINK_FACTOR** defaults to 1.0.

- **SOIBACK=**<grounded_voltage_source>

  This option applies to the SOI model only (BSIMSOI3 model). Instances of such models can accept 4 or 5 pins in the netlist. When the **SOIBACK** option is specified and the MOS device is defined as a 4 pin device, then the 4th pin is the internal body of the device. This option defines a voltage source allowing Eldo to translate a 4-pin SOI instance into a 5-pin SOI instance. The backgate will be inserted and connected automatically to the voltage source defined by the **SOIBACK** option.

- **SPMODLEV**

  Beginning Eldo v6.5 the SP model was enhanced with analytical derivatives. Analytical derivatives replace the numerical derivatives and improves the speed of the model. By default Eldo uses the new implementation. To select the old model implementation, set the Eldo option **SPMODLEV** for backward compatibility.

- **TMAX|TMIN**=VAL

  Some models contain self-heating effects, such as VBIC or Hicum models. For such models, when self-heating is active, the temperature of the device becomes an unknown to the system. In order to prevent possible overflow in model evaluation, device temperature is limited by Eldo, and by default cannot exceed 1000K, or go below 0K. These values can be overwritten using options **TMIN**=VAL or **TMAX**=VAL.

- **USEDEFAP**

  If the model parameter **ACM** is set to 2 or 3, then **AD**, **AS**, **PD** and **PS**, when unspecified in the instance command, are computed from **W**, **L** and **HDIF**, regardless of what values are given to **DEFAD**, **DEFAS**, **DEFPD** and **DEFPS**. If you wish to use **DEFAD**, **DEFAS**, **DEFPD** and **DEFPS**, option **USEDEFAP** has to be set.

- **WARNING_DEVPARAM**

  Forces Eldo to print a warning instead of an error when an unknown parameter is specified on a device instance. For example:

  ```
  ERROR  254: OBJECT "M1": Unknown parameter MULU0
  ```

  will be replaced by:

  ```
  Warning 209: OBJECT "M1": Parameter ignored MULU0
  ```

- **WARNMAXV**=VAL

  Returns a warning if a Voltage on a node is higher than VAL. This applies to DC analysis only.

- **WL**

  Reverses the order of MOS length and width specification.

- **YMFACT**

  Allows the use of the device multiplier **M** in Y instantiations. This must be specified at the beginning of the netlist.

- **ZDETECT**

  When this option is set, it will enable Eldo to detect 'Z' states on A2D nodes where an RZ value has been specified either in the **.A2D**/**.model** card or with the option RZ=VAL. A 'Z' state is detected when the equivalent impedance of the A2D node exceeds the specified **RZ** value command.

Reduction Options

Refer to the chapter "Eldo Reduction" on page 1085.

- **REDUCE_KEEP_OUTPUTS**=**YES**|**NO**

  See "Reduction Options" on page 1097.

- **REDUCE_KEEP_NODE**=node_name

  See "Reduction Options" on page 1097.

- **REDUCE_KEEP_INST**=instance_name

  See "Reduction Options" on page 1097.

- **REDUCE_MAX_CAP**=value

  See "Reduction Options" on page 1097.

- **REDUCE_MAX_IND**=value

  See "Reduction Options" on page 1097.

- **REDUCE_MAX_RES**=value

  See "Reduction Options" on page 1097.

Note

For backward compatibility, the old (pre-2009.1 release) "**RC_REDUCE_***" options are still accepted for reduction. A warning message is displayed to inform you that this syntax is deprecated, suggesting to use **.REDUCE** commands instead for greater efficiency. If both syntax is found in a netlist then the newer **.REDUCE** syntax takes precedence.

Noise Analysis Options

- **FLICKER_NOISE**=VAL

  Used in Noise Analysis as a frequency dependent noise model selector. Default value is zero. Values 0, 1, 2, 3 are used. Same functionality as using the **FLKLEV** model parameter.

- **THERMAL_NOISE**=VAL

  Used in Noise Analysis as a temperature dependent noise model selector. Five values are used: 0, 1, 2, 3, 4. Same functionality as using the **THMLEV** model parameter.

  See the appropriate sections in the Device Models chapter for details of the device noise models used for each case for the above two noise analysis options.

- **IKF2**

  Specifies that the **ONOISE** value returned is in V^2/Hz, instead of $V/\sqrt{Hz}$.

- **JTHNOISE**=VAL

  Selects the equations for thermal noise in JFETs. The equations are as follows:

  JTHNOISE=0

  Default equation is used: $id = \frac{8kT}{3} \cdot gm$

  JTHNOISE=1

  If vds > vdsat $Sid = \frac{8kT}{3} \cdot gm$

  Else $Sid = 4kT \cdot (gm + gds)$

  JTHNOISE=2

  $$Sid = \frac{8kT}{3} \cdot (gm + gds) \cdot \left(\frac{3}{2} - \frac{Vdseff}{(2 \cdot VDSAT)}\right)$$

  where $Vdseff = min(Vds, VDSAT)$

- **NONOISE**

  By default, it is assumed that all devices contribute to the total output noise, unless this option is specified. Setting **NONOISE** assumes all devices to be noiseless.

- **NOISE_SGNCONV**

  Allows the user to change the sign convention used for the computation of the noise parameters **BOPT** and **PHI_OPT**.

Simulation Display Control Options

- **ACSIMPROG**

  This option displays in the terminal window the simulation progress (percentage) during an AC analysis. This can be useful to monitor long simulations.

- **DCSIMPROG**

  This option displays in the terminal window the simulation progress (percentage) during a DC analysis. This can be useful to monitor long simulations.

- **ENGNOT**

  Prints numerical values in engineering notation, with scaling factors (for example U, MEG). Applies only to data generated by the **.OP** command, to data corresponding to DC values, or to data corresponding to **.PRINT** statements, and **.EXTRACT** or **.MEAS** values. This is similar to option **INGOLD**, whichever is specified last takes precedence.

- **INGOLD**=VAL

  Controls the printing of double precision numbers in engineering or exponential format. VAL can be 0, 1 or 2. This is similar to option **ENGNOT**, whichever is specified last takes precedence.

  INGOLD=0
  Use engineering format: exponents are given from a single character with the same convention as that for the input file. However, the 1.0e6 is expressed as X rather than as MEG.

  INGOLD=1
  Fixed and exponential. Numbers between 0.1 and 999 are written without exponential format. Other numbers use the exponential SPICE 2G6 format.

  INGOLD=2
  Exponential SPICE 2G6 format. This is the default.

- **LOCAL_NOWARN**[[=]msgid]

  Suppress warning messages in a local context only. Similar in functionality to option **NOWARN** (globally suppress warning messages). By default, all warnings are displayed by Eldo. If specified in a file included with **.LIB** or **.INCLUDE**, then only the display of warnings issued in this file and its descendants will be suppressed. As a consequence specifying **LOCAL_WARN** in the top netlist is equivalent to option **NOWARN**.

  Specifying **LOCAL_NOWARN** with the message ID value means only warning msgid will *not* be printed by Eldo.

- **MAXTOTWARN**[=VAL]

  By default, all warnings are displayed by Eldo. This option limits the total number of warning messages displayed; only val number of warnings will be displayed. Default is 0, meaning this is disabled.

- **MAXWARN**=VAL | **MAXWARN**msgid=VAL

  By default, all warnings are displayed by Eldo. This option limits the number of warning messages displayed for each warning number. For example, **MAXWARN**=2 specifies that a maximum of two messages will be displayed for each intended warning.

  Specifying the message ID, `msgid`, is similar to above but only the maximum number of messages for warning `msgid` is explicitly specified, and therefore has precedence over the **MAXWARN** value. For example: **MAXWARN**125=2 specifies that no more than two messages of warning number 125 will be displayed.

- **MSGBIAS**[=VAL]

  Usually there will only be three messages mentioning that PMOS are connected to 0. Use the **MSGBIAS** option without specifying a value if you want all such messages, or specify a limit with `VAL`.

- **MSGNODE**=VAL

  Limits the number of node connection faults reported. Eldo reports four types of node connection faults as follows:

```
Warning 107: node "xxx": Less than two connections.
Warning 108: node "xxx": This node is a floating gate.
Warning 113: node "xxx": Not connected to any element.
  This node is removed from the netlist.
Warning 252: OBJECT "xxx": Self-connected object not created.
```

  If **MSGNODE**=0 then all connection fault warnings are displayed. By default, **MSGNODE** is set to 3, which means Eldo displays each type of connection fault for the first three nodes on which the fault is detected. If the number of nodes at which a fault is detected exceeds the number specified by this option, then the following warning message is issued:

```
Warning 29: Set .option MSGNODE=0 to receive all such warnings.
```

- **NOWARN**[[=]msgid]

  By default, all warnings are displayed by Eldo. Specify **NOWARN** without any value to suppress the display of all warnings. This option must be placed at the top of the design, just after the title line, otherwise the warnings will continue to be displayed.

  Specifying **NOWARN** with the message ID value means only warning `msgid` will *not* be printed by Eldo. Similarly, this option must be placed at the top of the design, just after the title line.

- **NUMDGT**=INTEGER_VAL

  Specifies the number of significant digits printed for numerical values written to the ASCII output file. Controls the print accuracy; the simulation accuracy is not affected. Applies only to data generated by the **.OP** command, to data corresponding to DC values, or to data corresponding to **.PRINT**/**.PRINTFILE** statements, and **.EXTRACT** or **.MEAS** values. Default is 5 digits.

- **PRINTLG**=VAL

  This option is used for printout purposes only: if number of items which appear in a **.PRINT** card exceeds **PRINTLG**, values are printed out in different tables. This is to prevent lines being too long in the ASCII output file, making reading of the file uncomfortable for the user. Default is 8.

- **VERBOSE**

  This option forces Eldo to display more detailed reporting with some information messages in the standard output terminal. Eldo will print hints about syntax which is valid but ignored if the appropriate analysis is not found in the netlist. For example:

```
Warning 10001: No optimization command has been found in the netlist.

As a consequence:
    1) .option OPSELDO_NETLIST is ignored
    2) .PARAMOPT are interpreted like .PARAM using the initial value,
       or ignored if no initial value has been specified.
    3) GOAL=MINIMIZE is ignored on measurement FOO
    4) GOAL is ignored on measurement FOO2
    5) UBOUND is ignored on measurement FOO3
    6) GOAL=MAXIMIZE is ignored on measurement FOO4

Warning 10002: COMMAND .MC has not been found in the netlist.
  As a consequence:
    1) .option DISPLAY_CARLO is ignored
```

- **WARN**[[=]msgid]

  Specify to cancel any **NOWARN**/**LOCAL_NOWARN** option specifications as soon as they are encountered. If a message ID, msgid, is specified then only warning msgid is enabled.

- **WBULK**

  Eldo would print out a warning about positive bias on NMOS bulk (or negative bias on PMOS bulk) only once, unless **WBULK** is set. When set, all warnings will be printed out.

Simulation Output Control Options

- **ACOUT**=VAL

  This option controls how expressions VX(a,b) and IX(a,b) are computed in AC analysis: X in this case stands for **DB**, **M**, **P** or **GD**. Only two values are allowed, 0 or 1. In -compat mode, **ACOUT** defaults to 1, else it defaults to 0.

  0

  VX(a,b) or **I**X(a,b) are computed from the complex value **v**(a)-**v**(b) or from **i**(a)-**i**(b).

  1

  VX(a,b) or **I**X(a,b) are computed from the complex value **V**X(a,0)-**V**X(b,0) or from **I**X(a,0)-**I**X(b,0).

- **ALTER_NOMINAL_TEXT**

  This option allows to attach a label to the nominal run. For **.ALTER**, the label follows the command, for example: **.ALTER** `<label>`. This text is printed in the *.aex* file or attached to the JWDB waveforms.

- **ALTER_SUFFIX**

  Change the naming convention for swept waves used in **.ALTER** statements to be switched between: *xxx* and *xxx_alter:XX*.

- **ASCII**

  Forces Eldo to store the results including the printing of the output curves in the ASCII output *.chi* file. The default behavior is *not* to store the results in the ASCII output file (that is, **NOASCII** is the default). This option is equivalent to the `-ascii` command-line flag of Eldo. The size of this file can increase significantly for large simulations when these results are printed inside it. Affects **.PLOT** and **.PRINT** commands.

- **ASCIIPLOT**

  Forces Eldo to plot the waves in the ASCII *.chi* output file. By default, wave information is only written to the binary output *.wdb* file, with print tables in the *.chi* file.

- **BLK_SIZE**=VAL

  Sets the total size of the buffer when using the 4.7 cou file format. With this cou file format, data is buffered. The format is k values for X-axis, followed by k values for wave 1, then k values for wave 2 and so on. The value of ‘k’ (that is, the size of the buffer) is computed by Eldo according to the number of waves. Specified in Octets. Default is 1e6 (that is, 1Meg).

- **CAPTAB**

  Prints out in the ASCIII *.chi* output file the capacitance values which are placed in the AC matrix. The term Cij corresponds to the derivative of the charge on node ‘i’ with respect to the voltage on node ‘j’. **CAPTAB** must be used in conjunction with a **.AC**, **.OP** or **.TRAN** command. Information will not be output when there is only a **.DC** command. The output in the ASCII file has the following format:

```
Node <i>
<j1> C = <val1>
<j2> C = <val2>
...
CFIX = <valfix> CVAR = <valvar> CTOT = <valtot>
```

  where:

  `val1` is the derivative of the charge on node ‘i’ with respect to the voltage on node ‘j1’, and so on.

  `valfix` is the total capacitance connected to node ‘i’ which is not dependent on other node voltages.

  `valvar` is the total capacitance connected to node ‘i’ which is dependent on other node voltages.

`valtot` is the sum of `valfix` and `valvar`.

- **`COLLAPSE_DSPF_OUTPUT`**`[=@|#]`

  When set, Eldo will regroup currents (I, Ix, Isub) and noise outputs according to the device or subckt instance specified in arguments. By default the DSPF character is set to @. (Character # can also be specified with this option.) For example, the equivalence for currents can be expressed with the SIGMA() operator:

  `Ix(X1.1)` is equivalent to `SIGMA(Ix(X1.1), Ix(X1@*.1))`
  `I(R1)` is equivalent to `SIGMA(I(R1), I(R1@*))`

  Whereas noise outputs are SQRT (sum (NOISE(...) ^2)):

  NOISE(X1) is equivalent to:
  SQRT (NOISE(X1)^2 + NOISE(X1@2)^2 + ... NOISE(X1@n)^2)

  Limitation: the printing of noise in the *.chi* file will still report the parasitic devices.

- **`CONTINUOUS_FFT`**

  Specifies the waveform viewer to represent the FFT wave in continuous mode instead of the default spectral representation. This modifies the *.swd* file only; FFT waveforms are still saved in the *.wdb* file as spectral. Can alternatively be specified with the (**`CONTINUOUS`**) keyword on the **`.PLOT`** command.

- **`DEFRMSNTR`**

  When a **`.NOISETRAN`** simulation is requested, transient analysis extracts by default return only nominal values, see ".NOISETRAN" on page 747. For backward compatibility (pre-v2008.2), specify this option for Eldo to return both RMS and nominal values. In such a case, **`.EXTRACT TRAN`** extracts (or extracts without any analysis type) will return both RMS and nominal values: RMS values computed by **`.NOISETRAN`**, nominal values from the nominal transient analysis. This only has an effect for **`.NOISETRAN`** simulation with multiple run **`MRUN`** specified.

- **`DISPLAY_CARLO`**

  Display all the updated values for all the Monte Carlo and Worst Case runs.

- **`DUMP_EXTRACT`**`=0|1`

  Save single datapoint extract results (for example, scalars) in the *.wdb* file, so that they can be compared between *.wdb* files.

  0
  : Default. Do not save single datapoint extract results in the *.wdb* file.

  1
  : Save single datapoint extract results in the *.wdb* file.

- **`DUMP_MCINFO`**

  Write a summary of the **`.EXTRACT`** distributions during Monte Carlo analysis to the *.aex* file and the file declared by the **`.EXTRACT`** parameter **`FILE`**`=FNAME`, if they are specified. The information is also written to the *.chi* file.

- **EMPTY_MCHISTO**

  Forces Eldo to create single point histograms for **.EXTRACT** commands which show no variations during a Monte Carlo analysis. By default no histogram is generated for such extracts. This option is similar in functionality to the **HISTO_ZERO** option.

- **EXTCGS**

  The default unit for **.EXTRACT** current values (I(object_name)) is Amperes (since Eldo v6.5_2). Set this option to specify mA as the default unit for backward compatibility.

- **EXTFILE**

  Enables the generation of a *.ext* or *.ext.wdb* file depending on the output format *.cou* or *.wdb*. Extraction results are by default saved inside the EXT folder in the main *.wdb* file. By default, the extraction results file (*.ext* or *.ext.wdb*) is not automatically created.

- **EXTMKSA**

  When set, the **.EXTRACT** values which are relevant to the transient-extract language are expressed in *mksA* (meter-kilogram-second-Ampere) units instead of *cgs* (centimeter-gram-second) units, that is, if **EXTMKSA** is set, a result such as:

  ```
  TPDUU(V(1),V(1),VTH=0.5) = 1.0001E+06 nS
  ```

  would be returned as:

  ```
  TPDUU(V(1),V(1),VTH=0.5) = 1.0001E-03
  ```

- **EXTMOD_GENWAVE**

  If this option is set, the extract-mode will use the run parameters from the input database to regenerate a *.wdb* file containing EXT folders only.

- **EXTRACT_EVAL_FINAL**

  When specified, **.EXTRACT** function **EVAL**(device_name,device_entity) will take into account the scale factor, option **SCALE**, if specified. The value returned will be the value computed by the Eldo parser, multiplied by the scale factor. When this option is not specified, the value returned by **.EXTRACT** function **EVAL**(device_name,device_entity) is the value computed by the Eldo parser, before any scale factor effect is applied. For example:

  ```
  .param p1 = 1u
  .param p2 = p1*2
  M1 ... w = p2
  .option scale = 3

  .extract eval(m1,w)
  ```

  By default, Eldo will return an extracted value 2u, that is, the value of p2. If option **EXTRACT_EVAL_FINAL** is set, Eldo will return an extracted value of 6u, that is, the value of p2 multiplied by **SCALE**.

- **EXTRACT_VECT_AXIS**=**INDEX**|**XAXIS**

  Controls the type of x-axis used for vector **.EXTRACT** waveforms in the *.wdb* output file. It applies only to "Transient Extraction Language Functions" on page 645 (those extracted at run-time) used in conjunction with the **VECT** keyword. The default value is **INDEX**, extract vectors are plotted versus an increasing abstract index. When set to **XAXIS**, extract vectors are plotted using the same x-axis as normal waveforms, and values are saved at the time they have been measured. For example:

```
.option EXTRACT_VECT_AXIS=XAXIS
v1 1 0 sin(0 1 1meg)
r1 1 0 1
.tran 1n 5u
.extract tran vect trise(v(1),vl=0.2,vh=0.8)
.end
```

- **HISTLIM**

  In case History has to be saved on FAS, **HISTLIM** would save data only if there is a significant change (based on **RELTOL**) between the last saved value and the current value. This is in order to conserve memory when running very long transient simulations.

- **HISTO_ZERO**

  Forces Eldo to create a histogram of **.EXTRACT** commands even if the range of variation is 0. By default, when the range is 0 (when the **.EXTRACT**/**.MEAS** is not affected by Monte Carlo changes), no histogram is generated. This option is similar in functionality to the **EMPTY_MCHISTO** option.

- **INPUT**

  Forces a list of the user-specified inputs present in the netlist to be written to the binary output file (*.wdb*) for checking purposes.

- **INFOMC**=filename

  Forces Eldo to dump the information in the specified filename instead of in the *.chi* file. This function emulates option **DISPLAY_CARLO**.

- **JWDB_ACTRAN_USE_TIME**

  Instructs Eldo to use the time value, instead of an absolute counter, to name the AC or NOISE folders which are created in the *.wdb* file during transient analysis. By default, they are named AC_1, AC_2, and so on. If this option is set, they will be named AC_20.000N, AC_50.000N, and so on.

- **JWDB_EVENT**

  Specifies the time interval in seconds for updating marching waveforms displayed in the EZwave viewer. Default value is 10 s.

- **JWDB_EXTENSIONS**=0|1

  When set to 1, enables modifications in the way results are stored inside *.wdb* files. The compound waveform is not created when **.MC** irun=X is used (since this is a single simulation). Default is 0.

- **JWDB_PERCENT**

  Specifies the simulation percentage interval for updating marching waveforms displayed in the EZwave viewer. Default value is 5%.

- **KEEP_DSPF_NODE**

  This option forces Eldo to keep the original **.PLOT**/**.PROBE** command if a node coming from the DSPF has the same name. It is not set by default because this node may be completely different than the original one. When a node is replaced by a parasitic net using a DSPF file, this node can be renamed or even removed from the design. This is why **.PLOT**/**.PROBE** commands referencing a node modified by a DSPF file generate a compound waveform regrouping the parasitic nodes.

- **KEEP_HMPFILE**

  Generates the *.hmp* output file for transient noise analysis results, see “.NOISETRAN” on page 747. The *.hmp* file, of COU format (a legacy format), contains results for each NOISE analysis. By default, it is not generated. If the **NONOM** parameter is specified, then this option is always ignored and the *.hmp* file is not generated, This is because only one run, the noisy run, is performed. This is written directly to the regular *.wdb* file, without the need for an extra *.hmp* file, which is only used to compute RMS values when the simulation is complete.

- **LCAPOP**

  Synonymous with the **CAPTAB** option. Prints out in the ASCII *.chi* output file the capacitance values which are placed in the AC matrix. The term Cij corresponds to the derivative of the charge on node ‘i’ with respect to the voltage on node ‘j’. **LCAPOP** must be used in conjunction with a **.AC**, **.OP** or **.TRAN** command. Information will not be output when there is only a **.DC** command. The output in the ASCII file has the following format:

```
Node <i>
<j1> C = <val1>
<j2> C = <val2>
...
CFIX = <valfix> CVAR = <valvar> CTOT = <valtot>
```

  where:

  `val1` is the derivative of the charge on node ‘i’ with respect to the voltage on node ‘j1’, and so on.

  `valfix` is the total capacitance connected to node ‘i’ which is not dependent on other node voltages.

  `valvar` is the total capacitance connected to node ‘i’ which is dependent on other node voltages.

  `valtot` is the sum of `valfix` and `valvar`.

- **LIMPROBE**=VAL

  Sets the maximum number of nodes that may be monitored via the **.PROBE** command. Default value is 10,000.

- **`LIST`**

  Causes a listing of the elements contained in a circuit netlist, together with the names of their pins, to be printed to the ASCII output (*.chi*) file. Grounded capacitors are not considered as elements.

- **`MAX_CHECKBUS`**`=ALL|val`

  This option allows the user to control the number of checkbus errors printed to the *.chi* file. Default value is 20. Specifying 0 means no errors are printed. `ALL` specifies all errors are printed (this can also be set by specifying `-1`). Also if the **`.CHECKBUS`** command has specified the **`LOCK`** parameter then all check points are printed whether they have passed or failed.

- **`MAX_DSPF_PLOT`**`=VAL`

  Sets the maximum number of DSPF interface nodes plotted. Default value is 100. This can be useful when a plot (or probe) in DC or TRAN is asked on a node which has been replaced by a DSPF parasitics network (**`.DSPF_INCLUDE`** command), then the plot of this node will be replaced by a plot of all the DSPF network interface nodes. Inside EZwave, the generated plots will be grouped inside an analog bus.

- **`MC_NOMINAL_OP`**

  Forces operating point (OP) results to be saved for the nominal Monte Carlo run. By default, **`.MC`** used without the **`ALL`** keyword disables option **`PROBEOP`**, option **`PROBEOP2`**, and PSF OP information.

- **`MEASFILE`**

  Enables the generation of a *.meas* or *.meas.wdb* file depending on the output format *.cou* or *.wdb*. Measurement results are by default saved inside the EXT folder in the main *.wdb* file. By default, the measurement results file (*.meas* or *.meas.wdb*) is not automatically created.

- **`NEWACCT`**

  Specifies that simulation display statistics are displayed in a better format. By default, this option is disabled. When this option is made active, the subsequent output will be generated with one item and one number allowed per line. This simplifies the writing of post-processors. Example:

```
Number of Input signals            3
Number of resistors                0
Number of floating capacitors      0
Number of inductors                0
Number of voltage sources          3
Number of current sources          0
Number of dependent sources        0
Number of diodes                   0
Number of BJT                      0
Number of JFET                     0
Number of MOS                      19
Number of SWITCHES                 0
Number of Transmission lines       0
Total number of elements           22
Number of equations                13
```

```
Number of non-zero terms            73
Percent Zeros                       5.680e+01
Number Newton iterations            178
Average number Newton iterations    2.000e+00
Number of accepted time steps       57
Number of rejected time steps       4
      due to LTE                    4
      due to newton to Newton       0
Evaluation of active devices        11864
```

- **NOASCII**

  Eldo sets this option by default. It forces Eldo to suppress the printing of the output waveforms in the ASCII output *.chi* file. The output file is still created, but the size is significantly reduced for large simulations when these results are not printed inside it. This option is equivalent to the `-noascii` command-line flag of Eldo. Affects **.PLOT** and **.PRINT** commands. Specify the **ASCII** option to force Eldo to store the results including the printing of the output curves in the ASCII output *.chi* file.

- **NOASCIIPLOT**

  This option forces Eldo to suppress the printing of the plots in the ASCII *.chi* output file. Information is written only in the binary output *.wdb* file.

- **NOBOUND_PHASE**

  Option to avoid the fold-down for the phase. When this option is set, the phase is not displayed in modulo 360 degrees. This option only applies to outputs of type *X*P() (for example vp(), ip() or wp()). It does not apply to outputs of another type, or to the internal representation of waveforms of type phase. Therefore, to avoid fold-down of phase for a computed waveform, it should be written in the form:

  ```
  .plot ac wp(computed_waveform)
  ```

- **NODCINFOTAB**

  Disables the printout of the DC node information in the ASCII output file. This can be useful to save disk space if the circuit is very large and the DC solution is not of interest.

- **NODE**

  Causes printing of a node table to the log (*.chi*) file. It contains a list of all elements in a circuit netlist, together with the total grounded capacitance value for each node.

- **NODEFRMSNTR**

  When a **.NOISETRAN** simulation is requested, transient analysis extracts by default return only nominal values, RMS values are not returned. Transient noise analysis sets this option by default, see “.NOISETRAN” on page 747. Specify option **NODEFRMSNTR** if only nominal values should be dumped. Use the analysis type **NTR** on the **.EXTRACT** command to select computation based on RMS.

- **NOEXTRACTCOMPLEX**

  The **.EXTRACT** command by default provides complex results information instead of only real values with the yval function. This can be disabled by setting this option. Complex results information is used by default instead of real values with the **.EXTRACT** yval function. For example, `.extract ac label=EX1 yval(v(out),10k)` would deliver the result:

```
* EX1 = 10.011 , -1.8
```

The general form is: real_part, imaginary_part.

- **NOMOD**

  Suppresses the print-out of the model parameters to the ASCII output (*.chi*) file.

- **NOOP**

  Suppresses the printing of operating point (OP) table information in the ASCII output (*.chi*) file when a **.AC** analysis is specified.

- **NOPAGE**

  This is synonymous to the option **LIST**.

- **NOSIZECHK**

  Eldo checks the size of the *.chi* file, and does not write to it if its size exceeds the total disk space available (Unix Only). The **NOSIZECHK** option disables this check.

- **NOSMKMCWC**

  SOA are computed and dumped for all runs triggered by **.MC** and **.WCASE** commands (by default beginning Eldo v6.8). This enables the user to see if SOA are violated or not during MC or WCASE runs. Specify this option to disable this, in which case information will be dumped only for the nominal run.

- **NOTRC**

  Suppresses the rewriting of the circuit description file in the ASCII output (*.chi*) file. Only the lines following its declaration in the netlist are suppressed. See also “.NOTRC” on page 752 to fully suppress the description from being rewritten.

- **NOTRCLIB**

  This option removes the printout of **.MODEL** and **.SUBCKT** included from library files via **.LIB** or **.ADDLIB** commands.

- **NOWAVECOMPLEX**

  Disables the generation of complex waves. By default, complex waveform information is only written to the saved windows file (*.swd*) with JWDB format output. This option forces Eldo to write complex waveform information to the waveform database (*.wdb*) as well as the saved windows file. This can be used to restore the functionality of Eldo versions prior to v6.3_2. This can also be specified using the `-jwdb_nocomplex` flag when invoking Eldo.

- **NOXTABNOISE**

  X instances that do not contain any other X instances are considered as Eldo primitives, and are taken into account in the noise table that is displayed in the ASCII output files. This is particularly useful in the case of designs where devices are modeled by subcircuit rather than by Eldo primitive. Such X instances will then appear in the ASCII table as if Eldo primitives had been used. This can be disabled with the **NOXTABNOISE** option.

- **OPALLDC**

  Creates an "OPERATING POINT INFORMATION" section in the *.chi* file for each point of a DCSWEEP analysis. The total power dissipation is also reported in the standard output for each point. It does not affect other output files.

  The presence of a **.OP** command is not necessary to enable the printing in the *.chi* file when option **OPALLDC** is set.

- **OPTYP**=VAL

  Used to change the way Operating Point information related to MOSFETs is displayed in the ASCII output file. Default: **optyp**=1. See ".OP" on page 755.

- **OUT_RESOL**=VAL

  Save data in binary waveform output file only if the time increment in the output file is at least the value of **OUT_RESOL**.

- **OUT_SMP**=VAL

  This can be used as an alternative to the **FREQSMP**/**TIMESMP** with **OUT_STEP** options. With **OUT_SMP** Eldo expects a frequency value to be specified whereas a time interval is expected with **OUT_STEP**.

- **OUT_STEP**

  Used in conjunction with option **FREQSMP**=VAL. Forces Eldo to save into the binary waveform output file only the timesteps in 1/val intervals as defined by **FREQSMP**. All intermediate timesteps, although calculated, will not be saved. The results in this output file (using **.OPTION FREQSMP**=val **OUT_STEP** in the netlist) are close to those using option **FREQSMP**=val in the netlist, but not to the output file without option **FREQSMP** in the netlist.

  It is possible to assign expression parameters to **OUT_STEP**. For example:

  ```
  .PARAM p1 = 1k
  .OPTION OUT_STEP = p1
  ```

- **PARAMETRIC_ACTRAN**

  Modifies the way Eldo treats AC or NOISE extracts which are performed during transient (AC + TRAN + OP time). By default Eldo generates a new extract for each time value. When this option is set, Eldo will not generate new extracts but will consider the extract has a parametric waveform with time for sweeping variable.

- **POST**[=1|2]

  Specifying this option without any value is equivalent to **.PROBE V**, but in addition, current values of grounded voltage sources are also displayed.

  When **POST**=1, additional results files are generated as binary files.
  When **POST**=2, additional results files are generated as ASCII files.

- **POST_DOUBLE**

  Save values in output files using double precision. By default output files generated by option **POST**=1|2 are saved using simple precision variables (float). This can lead to accuracy problems in certain cases.

- **PRINT_ACOP**=0|NO|1|YES

  This option will limit the amount of operating point information produced in the output file, when a **.AC** analysis is specified.

- **PRINTFILE_STEP**=VAL

  Defines a default global value for all **.PRINTFILE** commands that do not have a specific STEP value.

- **PRINTFILE_FREQ_STEP**=VAL

  Defines a default global value for all time-based **.PRINTFILE** commands that do not have a specific STEP value.

- **PRINTFILE_TIME_STEP**=VAL

  Defines a default global value for all frequency-based **.PRINTFILE** commands that do not have a specific STEP value.

- **SIMUDIV**=VAL

  Specifies how many times status information will be printed out during simulation. For example, **SIMUDIV**=10 causes a print out after each 10^{th} of the simulation, for example each 10^{th} of `tstop`. Status information recorded includes the elapsed CPU time, estimated total CPU time, percentage of simulation done, plus any data selected using the **STAT** option. Can also be used independently of **STAT**. Cannot be used in conjunction with **TIMEDIV**. Default is 10. Set **SIMUDIV** to 0 to remove the print out.

- **STAT**=VAL

  Used to set debug levels for simulation. Four values are allowed: 0, 1, 2, 3.

  0
  : Default. Specifies no statistics are to be recorded.

  1
  : Reports **NEWTON**/**OSR** partitioning in the *.chi* file and to the terminal. Writes a trace of parameter and object updates in *.chi* log file. Simulation time data is written to the *.chi* file when **SIMUDIV** or **TIMEDIV** are used.

  2
  : As level 1 plus: a node list is written to the *.chi* file.

  3
  : As level 2 plus: parameter and object update tracing is also listed to the terminal (`stdout`). The number of rejected/accepted timesteps is reported when **SIMUDIV** or **TIMEDIV** are used.

- **NOSTATP**

  Disables the print out of parameter values when **STAT** option is set to a non-zero value (**STAT** is then used only for partitioning information).

- **TIMEDIV**=VAL

  Works in conjunction with **STAT**. This option is equivalent to **SIMUDIV** except VAL now specifies a CPU time interval. Status information will be printed out after every VAL minutes of elapsed CPU time. This may significantly slow down simulation. Cannot be used in conjunction with **SIMUDIV**.

- **TEMPCOUK**

  By default, all the temperatures are plotted in Celsius in the binary output file (*.wdb*). Setting this option causes the temperatures to be plotted in Kelvin.

- **VBCSAT**=VAL

  Default is 0. Sets the 'region of work' that is displayed on the **OP** table for BJT as follows:

  ```
  if(VBC > VBCSAT)
     if(VBE > 0) "SATURATION"
     else "INVERSE"
  else
     if (VBE > 0 ) "ON"
     else "OFF"
  ```

- **VXPROBE**

  Adding this option with **.PROBE V** forces Eldo to dump ground and subcircuit node voltages to the output file.

- **WRITE_ALTER_NETLIST**

  When specified, for each **.ALTER**, the altered netlist used by Eldo is written in the *.chi* file.

Optimizer Output Control Options

- **OPSELDO_ABSTRACT**

  Generates a summary table of simulations containing parameter and extract values for each run.

- **OPSELDO_DETAIL**=[NONE|ALL]

  If this option is set to NONE, only the last run and the nominal run will be stored in generated files (*.wdb*, *.aex*) and no other simulation information will be displayed. When set to ALL, simulation information for all runs will be stored. Default is NONE.

- **OPSELDO_DISPLAY_GOALFITTING**

  Displays the intermediate goals when splitting DC measurements during optimization. Each **.DATA** point is optimized as an independent goal.

- **`OPSELDO_FORCE_GOALFITTING`**

  Forces Eldo to split DC fitting extracts in independent goals even if data points are correctly ordered (as opposed to **`OPSELDO_NOGOALFITTING`**). It can also split AC fitting extracts.

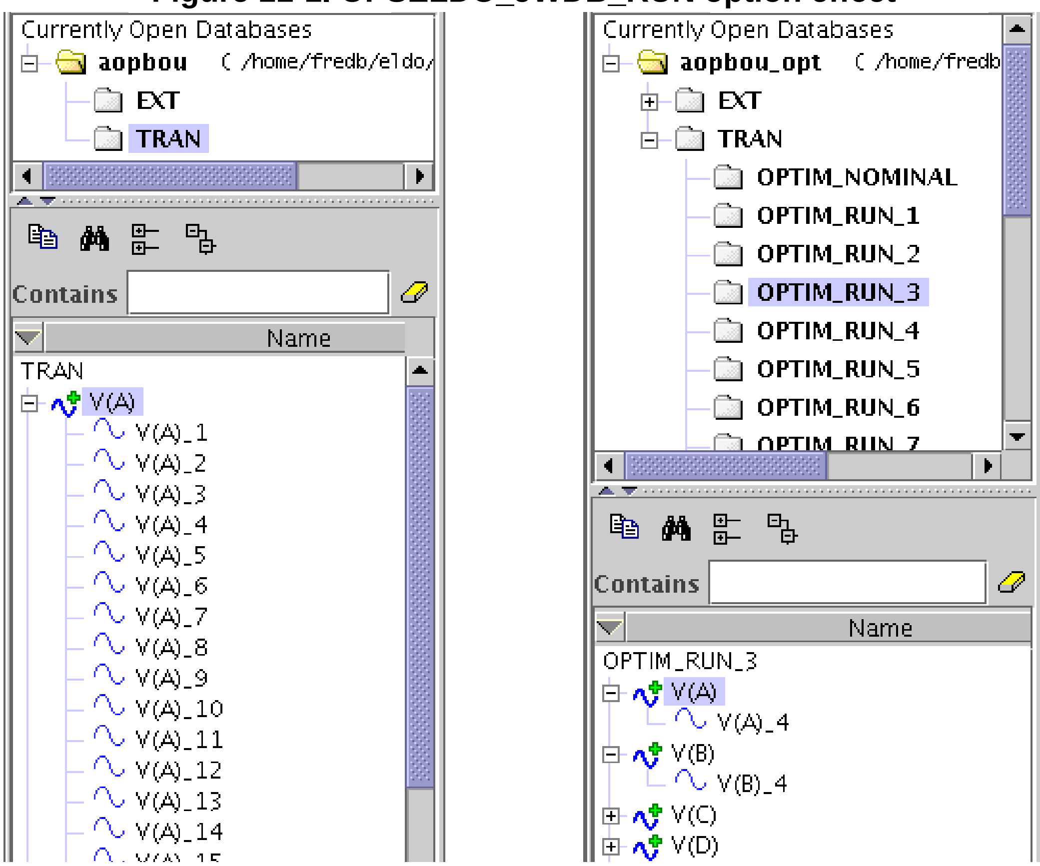
  Disables the splitting of DC measurements during optimization. By default, each **`.DATA`** point is optimized as an independent goal.

- **`OPSELDO_JWDB_RUN`**

  Organizes the Waveform List of EZwave in a different way allowing easy access to results for different optimizer runs. This can also be specified using the `-opseldo_jwdb_run` flag when invoking Eldo.

  The effect of the **`OPSELDO_JWDB_RUN`** option can be seen in Figure 11-1; the image on the left is without the option, the image on the right is with the option specified.

Figure 11-1. OPSELDO_JWDB_RUN option effect



- **`OPSELDO_NETLIST`**

  Generates a netlist modified from the original input file, which contains the optimized parameter values but also every parameter set under #ifdef statements.

- **OPSELDO_NO_DUPLICATE**

  When a **.PARAMOPT** is declared twice Eldo will generate both a warning and an error with this option specified. By default, double definition of **.PARAMOPT** is allowed with only warnings generated.

- **OPSELDO_NOGOALFITTING**

  Disables the splitting of DC measurements during optimization. By default, each **.DATA** point is optimized as an independent goal only if they are not correctly ordered.

- **OPSELDO_OUTER**

  Allows a reverse behavior of optimization and sweep simulations (**.TEMP**, **.DATA** or **.STEP**). A full optimization will be performed for each set of sweep parameters.

- **OPSELDO_OUTPUT**

  This option controls results of optimization (Opsim or bisection method).

  OPSELDO_OUTPUT=0
  Print results in a simplified format: parameter name, goal, optimized value

  OPSELDO_OUTPUT=1
  Print results using simplified format (as for **OPSELDO_OUTPUT**=0) and generate a *.ops0* file using the same format.

  OPSELDO_OUTPUT=2
  Print results in a detailed format: parameter name, minimum value, maximum value, weight, goal, optimized value

  OPSELDO_OUTPUT=3
  Print results using detailed format (as for **OPSELDO_OUTPUT**=2) and generate a *.ops0* file using the same format.

- **RESET_MULTIPLE_RUN**

  This option is used internally by Eldo to disable multiple-run analyses (**.STEP**, **.TEMP**). It is only used in the files generated by Eldo to replay some optimization runs. Parameters are assigned values and multiple-run analyses are disabled.

Caution
The behavior cannot be predicted if this option is specified by the user instead of by Eldo.

File Generation Options

- **AEX**[=0|1|2]

  Forces Eldo to dump the results of **.EXTRACT** or **.MEAS** commands into a file *filename.aex*. The values are still also written into the *.chi* file. This is the default. See also option **NOAEX**.

  AEX=0
  Equivalent to option **NOAEX**.

AEX=1
Equivalent to option **AEX**.

AEX=2
Equivalent to option **AEX** with the header modified to report the version number of Eldo.

- **ALIGNEXT**

  Specifies that Eldo will write aligned **.EXTRACT** results in the *.aex* file (aligned using space characters). By default the results are not aligned. Remember, the *.aex* file is only created if the option **AEX** is activated.

- **ASCII**=VAL

  If VAL is set to 0, the ASCII output file generation is terminated. If VAL is set to 1, the ASCII output file is created. The default value is 1. This option does not have the same meaning as the option "ASCII" on page 998.

- **COU**

  Used to generate output in binary Cou format. This can also be specified using the invoke command `-gwl cou`.

- **CSDF**

  Used to generate output in CSDF format. CSDF is the Common Simulation Data Format. This can also be specified using the invoke command `-gwl csdf`.

- **DUMP_FILE_LIST**

  Generates a file containing all the files opened by Eldo (except temporary Eldo files) for a simulation. Useful for packaging an Eldo testcase. If a filename is not specified, then the netlist name is used with the extension *.files_dump*. If a relative path is specified then the file will be generated from the directory specified in `-outpath`.

  The generated file contains all the absolute names of the opened files and is sorted by path and then by name (case ignored). Only one file is generated even if there are **.ALTER** statements or multiple netlists in the source file.

  This option is equivalent to the `-dump_file_list` command-line flag of Eldo.

- **FSDB**

  Used to generate output in FSDB format for nWave. This can also be specified using the invoke command `-gwl fsdb`.

- **FSDB_SINGLE_FILE**

  Removes the analysis name from the FSDB output filename and from the waveform hierarchy. By default, FSDB output files are named using the analysis, *<netlist>.<analysis_name>.fsdb* because they can only contain one analysis. The name of the analysis also appears in the waveform hierarchy like folders in JWDB.

  If multiple analyses are performed in the netlist, the FSDB file will be replaced from one analysis to another.

- **INFODEV**=file_dev

  Eldo will generate a file `file_dev` that contains all of the R/L/C/M/B/D/J device names with their corresponding parameter values evaluated.

- **INFOMOD**=file_mod

  Eldo will generate a file `file_mod` that contains all of the **.MODEL** commands with their parameter values evaluated.

- **ITRPRT**

  Forces Eldo to dump in the ASCII output table all the points which have been computed (internal time points), instead of those computed from the time interval specified in the **.TRAN** command. Note that with this option set, the ASCII output file can be huge.

- **JWDB**

  Used to generate output in JWDB format (extension *.wdb*). This can also be specified using the invoke command `-gwl jwdb`. See also option **NOJWDB**.

- **NOAEX**

  Suppresses the **AEX** option and hence cancels the creation of *filename.aex*. The values are only written into the *.chi* file. See option **AEX**.

- **NOJWDB**

  Specifying this option will disable the generation of the output in JWDB format. This can also be specified using the `-nojwdb` flag when invoking Eldo. See also option **JWDB**.

- **NOCOU**

  Suppresses the generation of the binary monitored results (*.cou*) file. See also option **COU**.

- **NOIICXNAME**

  Used to restore the full name display in the *.iic* file. (This was the default behavior in releases prior to v6.5.) By default, the saved simulation run file (*.iic*) format compresses the hierarchical names, in order to reduce the size of the file. With this option, the file produced can become very large when simulating large circuits with lots of hierarchy because the node names are stored flat.

- **NOCKRSTSAVE**

  By default, if the same file name appears in a **.SAVE** command and in a **.RESTART** command, and if conditions for reading and writing the file are not the same, then Eldo issues an error. This is in order to prevent Eldo unexpectedly overwriting restart files which take a long time to create. **NOCKRSTSAVE** disables that check.

- **NOPROBEOP**

  When running Eldo in the ST mode, this will remove creation of the PROBEOP file.

- **NSAFILE_FORMAT**=**COU**|**WDB**|**NONE**

  By default, the noise response of devices output by **.OPTNOISE** are merged into the NOISE folder of the *.wdb* file. Use this option to control the output generated. Set to **COU** for backward compatiblity (pre-2009.2) behavior to create a separate *<netlist>_nsa.cou* file. Set to **NONE** to disable the creation of this file if you are only interested in the table inside the *.chi* file. Default is **WDB**.

- **OUT_REDUCE**

  Used to reduce the accuracy of output waveforms by performing a slope based comparison of output points. This is useful to reduce the number of points in case you have both constant (or slow varying) waves and varying waves in the same file. The comparison works by searching for groups of three points which are aligned. The middle point is ignored if the following condition is met:

  $$|current\_slope - previous\_slope| < OUT\_ABSTOL + OUT\_RELTOL \times |current\_slope|$$

  Where **OUT_ABSTOL** and **OUT_RELTOL** are options which can also invoke the waveform reduction comparison.

  As soon as this option is set, output waveforms are stored asynchronously that is, each wave will have it's own X values. Thus the size of the *.wdb* waveform database file may decrease or increase depending on the variations of waveforms. This option is useful if output waveforms contain constant signals, pulses, or slow varying signals. For example, power supplies which are constant require only two points at tstart and tstop.

- **OUT_ABSTOL**=value

  Used to reduce the accuracy of output waveforms and set the absolute tolerance value. Default is 1.0E-5. See **OUT_REDUCE** for details.

- **OUT_RELTOL**=value

  Used to reduce the accuracy of output waveforms and set the relative tolerance value. Default is 1.0E-6. See **OUT_REDUCE** for details.

Caution

Simulation time could increase when specifying options **OUT_REDUCE**, **OUT_ABSTOL** or **OUT_RELTOL**. They are not performance options.

- **PROBE**

  When -compat is set, then the Eldo **.PROBE** command is emulated, that is, all node voltages are dumped in the binary output file (*.wdb*). In order to have only those items specified in **.PLOT**/**.PROBE** to be dumped in the output file, add option **PROBE**.

- **PROBEOP**

  This option will be active only if **.OP** is explicitly required. By default, **.AC** will not create the probeop file. The **PROBEOP** option is used to write in a file the operating point information displayed in the chi file in a format easier to read for post-processors. If **PROBEOP** is set, the name of the file is *probeop*.

- **PROBEOP2**

  Eldo will generate two files: <*circuit*>*.op*<*x*> and <*circuit*>*.mapop*. For <*circuit*>*.op*<*x*>, *x* is the index of the run, it contains OP information for the current run. This file is an ASCII file. The first line of the file contains the ALTER index (if any), the MC index (if any), as well as the temperature values and the step value if **.STEP** is specified. The content of this file is similar to that created when option **PROBEOP** is used, but some additional information is dumped: temperature, node information, and current out of X leaf-instances (leaf-cells are instances which do not call any other X instances). For <*circuit*>*.mapop*, each line contains an index, corresponding to the index of the *.op* file, followed by the same information which can be found on the first line of the corresponding *.op* file. This mapping file facilitates the search of a specific configuration, rather than having to visit the first line of each *.op* file.

- **PROBEOPX**

  Force Eldo, during the OP analysis, to display the current which flows in each pin of each subcircuit X instance. This command will be active only if **.OP** is explicitly required.

- **PSF**

  Used to generate output in Cadence format. This can also be specified using the command line flag `-gwl psf`. The list of directories and files created by the PSF format will be printed at the end of the *.chi* file. For example:

```
==== Files and directories created by the PSF output format ====
/home/user/test/analysisInst
/home/user/timeSweep
/home/user/logFile
/home/user/test/variables_file
```

- **PSF_NODEVICE_NOISE**

  Disables the saving of the individual noise contributions per device, and instructs Eldo to keep only total noise in the PSF noise file. This provides backward compatibility with versions of Eldo prior to v6.10_1. The noise contributions of all devices are saved in a PSF file named noise by default.

- **PSF_SCALARDC**[=0|1]

  Selects saving DC values in scalar PSF format or sweep PSF format. **PSF_SCALARDC**=1 selects scalar PSF format output. This is automatically selected if option **PSF_WRITE_ALL** is specified. **PSF_SCALARDC**=0 is the default for backward compatibility.

- **PSF_VERSION**=1|2

  Selects between two versions of the PSF output module in Eldo. **PSF_VERSION**=2 (default) selects the module introduced in Eldo v6.9; **PSF_VERSION**=1 selects the output module available prior to Eldo v6.9. (Both modules will output the same PSF output format: PSFversion 1.00.) With the newer output module selected, extended options **PSF_FULLNAME** and **PSF_ALL_FILES** become available. By default, Eldo will not enable these two options to stay compatible with Artist Link. The newer module makes it possible to generate files larger than 2Gb for TRAN and TSST analysis.

- **PSF_FULLNAME**

  Instructs Eldo to save all waveforms in the output file using their complete names. In default mode, V(OUT) is saved as OUT. Using this option allows to save phase noise, harmonic waveforms, and noise circles. Output files keep the usual PSF naming convention: time-domain files are called *timeSweep* and frequency-domain files are called *frequencySweep*. There is only one file per analysis domain.

- **PSF_ALL_FILES**

  Alias of **PSF_WRITE_ALL**.

- **PSF_WRITE_ALL**

  This option overrides the rule of creating one file per analysis domain (usually *timeSweep* and *frequencySweep*). It instructs Eldo to use the exact analysis name for the output file. For example **.SSTNOISE** analysis creates a "noise" file by default. With this option, the output file will be named *sstnoiseSweep*. It also enables the saving of all waveforms in the output file (using .H() extension for RF outputs), and the activation of **PSF_SCALARDC** option to save the DC output values in scalar PSF format.

- **PSFASCII**

  Used to generate PSF format data in ASCII. This can also be specified using the invoke command `-gwl psfascii`.

- **SAVETIME**=VAL

  Creates a *.sav* file which saves the context of a simulation every `VAL` hours of CPU time used in a transient analysis. If the system crashes during a run, the simulation may be restarted from the last saving time.

- **WDF**

  Used to generate output in FSDB format for nWave. This can also be specified using the invoke command `-gwl fsdb`.

Mathematical Algorithm Options

For more details on these options, please refer to the Speed and Accuracy and Integral Equation Method (IEM) chapters.

- **TRAP**

  Resets the integration method to "Trapezoidal" from "Backward Euler." Useful in interactive mode only. Related parameters: **SMOOTH**.

- **SMOOTH**

  When the "Trapezoidal" integration method is used, oscillation may occur on some output curves. This **SMOOTH** option forces Eldo to display at the mid-point of the computed time intervals, this operation acts as a filter and oscillations due to **TRAP** are removed.

- **BE**

  Forces “Backward Euler” integration to be used. “Trapezoidal” integration is used by default.

- **GEAR**

  Forces “Gear” integration to be used. Trapezoidal integration is used by default. Related parameters: **MAXORD**, **CHGTOL**, **NGTOL**, **FLUXTOL**, **RELTRUNC**. Take care with the gear algorithms as you may potentially miss some oscillations in circuits.

- **GNODE**

  Assists DC convergence for **.RAMP DC** and option **GRAMP**. **GNODE** is the conductance which is applied between each node and ground. The value input by the user is the conductance value to be used as the first **RAMP** point. Eldo will reduce the value down to 0 for the final **RAMP** point, when conducting a **RAMP** analysis.

- **METHOD**=**GEAR**

  See the option **GEAR** above (used only for compatibility with SPICE).

- **MAXORD**=VAL

  Used in conjunction with option **GEAR**. Specifies the maximum order of the Gear integration method and must be in the range 1 to 6. The value 1 is equivalent to the **BE** option. Default value is 2.

- **NEWTON**

  Forces Eldo to use only one Newton block on the design. This is the default algorithm.

- **NODEFNEWTON**

  Forces Eldo to use OSR by default, instead of Newton. See option **NEWTON**.

- **IEM**

  Forces Eldo to use only one IEM block on the design.

- **BLOCKS**=**NEWTON**

  Forces Eldo to use OSR on the whole circuit and Newton-Raphson algorithm on tightly coupled sub-blocks. If the whole circuit was tightly coupled, then Eldo will automatically switch to **NEWTON**, after issuing a message.

- **BLOCKS**=**IEM**

  Forces Eldo to use OSR on the whole circuit and IEM algorithm on tightly coupled sub-blocks. If the whole circuit was tightly coupled, then Eldo will automatically switch to **IEM**, after issuing a message.

- **ANALOG**

  Forces Eldo to create only one Newton block for the whole circuit and adjusts **EPS** to a maximum value of 1.0×10^{-4} in order to achieve higher accuracy results.

- **DIGITAL**

  When the OSR algorithm is selected, this forces nodes which are not in **ANALOG** blocks and which are connected to MOSFET, grounded capacitors, current sources or floating capacitors of less than **CAPANW** to be solved by OSR, all other nodes being treated by Newton-Raphson. This option is useful for large MOS circuits with extracted parasitic coupling capacitances.

- **OSR**

  Forces Eldo to use the OSR (One Step Relaxation) iteration technique whenever possible, that is, for MOS loosely coupled circuits. Related parameters: **DIGITAL**, **CAPANW**.

- **NORMOS**

  This option allows the removal of access resistances in analog blocks if low precision simulation is required. For circuit blocks simulated using OSR at a relatively low precision level, MOS access resistances are not treated as separate elements. Their effect is taken into account during MOS model evaluation in an approximate manner. It is sometimes possible for analog blocks simulated using either Newton Raphson or IEM to have these devices (that is, MOS transistor with an approximate access resistance effect). This happens for example when the circuit is dynamically re partitioned during simulation for convergence problems. Analog blocks (solved by Newton or IEM) should maintain a minimum level of precision hence access resistors should be created as new elements.

  Hence, specifying **NORMOS** selects the mechanism that allows dynamic creation/removal of these devices depending on the algorithm used: removal for OSR and creation for both Newton and IEM.

- **PSTRAN**

  Forces Eldo to use the PSTRAN (PSeudo TRANsient) algorithm prior to any other convergence aid. This algorithm is one of the DC convergence algorithms that are automatically used by Eldo when the standard DC algorithm fails to converge.

- **DPTRAN**

  Forces Eldo to use the DPTRAN algorithm prior to any other convergence aid. This algorithm is one of the DC convergence algorithms that are automatically used by Eldo when the standard DC algorithm fails to converge.

- **NODCPART**

  Do not insist on DC partitioning in DC analysis. In case of convergence difficulties, immediately go to the convergence aid algorithm.

- **DCPART**=VAL

  Here, VAL is the number of time partitions that will occur, the default is 5. **DCPART**=0 is equivalent to option **NODCPART**.

- **CSHUNT**

  Add a capacitance between each node and ground. Default is 0. This can be used to eliminate oscillation problems caused by numerical noise or high frequency oscillation.

- **GSHUNT**

  Add a conductance between each node and ground. Default is 0. This can be used to eliminate oscillation problems caused by numerical noise or high frequency oscillation.

Mixed-Mode Options

- **D2DMVL9BIT**

  Enables direct connection between HDL-A ports of type BIT with mixed-mode nodes connected to Eldo via convertors of type MVL9 (or `std_logic`).

  Further information can be found in “.A2D” on page 528 or “.D2A” on page 577.

- **DEFA2D**=`model_name`

  Eldo will automatically insert an implicit A2D convertor when needed. Without this option specified, it is normally necessary to specify an explicit A2D between ANALOG nodes and HDL-A ports.

  Further information can be found under “Option DEFA2D” on page 536.

- **DEFD2A**=`model_name`

  Eldo will automatically insert an implicit D2A convertor when needed. Without this option specified, it is normally necessary to specify an explicit D2A between ANALOG nodes and HDL-A ports.

  Further information can be found under “Option DEFD2A” on page 584.

- **DEFCONVMSG**

  Forces Eldo to print out in ASCII output files the convertor information related to the **.A2D** and **.D2A** which are added when options **DEFA2D** and **DEFD2A** are specified.

- **DYND2ALOG**

  Enables the threshold voltages VTHI and VTLO to be calculated dynamically for digital macromodels using actual values of the high and low bias. The bias values, VHI and VLO are computed using two extra pins defined by the user in the digital model instance. The value specified on the first pin provides the high voltage digital output value (VHI) and the value specified on the second pin provides the low voltage digital output value (VLO). The two extra pins are placed after the list of regular pins and before the model name in the model instance syntax, see “Digital Macromodels” on page 447 for more information. This option must be placed at the top of the design, just after the title line.

  The *active* VTHI and VTLO threshold values are also computed dynamically using the following equations:

```
VTHI_ACTIVE = VLO + VTHI*DV
VTLO_ACTIVE = VLO + VTLO*DV
```

  VTHI and VTLO are the values specified in the **.MODEL** command defining the digital macromodel or in the macromodel instance, VLO is the low output voltage value given on the second extra pin defined by the user, and DV is the voltage difference given by VHI - VLO.

- **DYND2ALOG2**

  Specifying this option activates the already existing option **DYND2ALOG** for dynamically calculating threshold values of digital macromodels, and also changes the way the thresholds are detected.

  For the detection of the rising edge, and in **DYND2ALOG2** mode, Eldo checks for V > VTH and VLAST <= VTH_LAST where V and VTH are the voltage and the threshold value at the current time point, and VLAST and VTH_LAST the same entities but at the previous time step.

  If **DYND2ALOG** is set, Eldo will check for: for V > VTH and VLAST <= VTH, which is also the formula used in case none of the options **DYND2ALOG2** and **DYND2ALOG** are set.

  This change of the expression has a major impact in the threshold detection for the case the pin which is used to carry the voltage used as the threshold value is also an input pin: the output of the small circuit will change depending on the option set.

- **MIXEDSTEP**=**FREE|LOCKED**

  Used for controlling time-step synchronization between Eldo and a digital solver (ADVance MS or Verilog-XL). This option can take two values: **FREE** or **LOCKED**. Default is **FREE**.

  When **MIXEDSTEP** is **FREE**, simulation is usually faster, but may fail on some rare cases with a message inviting the user to re-run the simulation with **MIXEDSTEP**=**LOCKED**.

  When **MIXEDSTEP** is **LOCKED**, the analog kernel will stop on each digital event even if these digital events have no immediate impact on the analog side. This mode of simulation is very robust, but is usually slower except in cases where most of the digital events have an immediate impact on the analog side.

- **PARTVDD**=`node_name`

  Instructs Eldo/ADMS that the specified node can be accepted as a boundary node between ADMS (Eldo) and ADiT even if it does not conform to the partitioning rules. Using this option can significantly reduce the simulation time, so must be used with caution.

- **PARTVDD_AMS_ALL**

  Instructs ADMS to put the option **PARTVDD** on all nodes connected to a VHDL-AMS model. The partitioner will consider all the nodes connected to a VHDL-AMS model to be low coupling nodes for partitioning. This will lead to more devices being simulated by ADiT. A list of the nodes that the option has been set on will be displayed. Using this option can significantly reduce the simulation time, so must be used with caution.

- **PARTGATE_AMS_ALL**

  Forces the partitioner to cut on Questa ADMS ports. This option is similar to **PARTVDD_AMS_ALL** but has the opposite behavior. That is, the option instructs ADiT that all nodes connected to VHDL-AMS models can be accepted as boundary nodes between ADiT and Eldo, even if they do not conform to the partitioning rules.

- **FS_PARTITIONING**=level

  Specifies different algorithms to be used for the partitioning rules used by Eldo and ADiT in white-box mode. There are six levels of partitioning, numbered from 5 (most partitioning) to 0 (least partitioning). The default is 4.

  The partitioner defines the objects that will be simulated by Eldo and ADiT. The basic principle of this partitioner is to modify the partitioning made by the user in order to reduce, as much as possible, the coupling level between partitions.

  Please refer to Partitioning Options in the *Questa ADMS User's Manual* for further information.

- **FS_PARTITION_DEBUG**[=val]

  Causes Eldo (ADMS) to display the first defined number of devices which will be simulated on the Eldo side while they are partitioned inside the netlist (using **.OPTION** or **.PART** commands) to be simulated by fast SPICE (ADiT). The reason why the device will be simulated with Eldo is also reported. If no value is specified, the default is 10.

  Please refer to Partitioning Options in the *Questa ADMS User's Manual* for further information.

- **FS_SOLVE_AMS_NODES**

  Instructs ADMS to force all nodes directly connected to VHDL-AMS nodes to be solved by ADiT.

- **NO_FS_VA**

  Forces Verilog-A models to be sent to Eldo instead of ADiT.

- **VAMODEL**=mod_name

  Used in conjunction with the **.HDL** command. See ".HDL" on page 678 for further information. The Verilog-A module name specified with this option has higher priority than a subcircuit of the same name when both exist. Only one module name can be specified on each **VAMODEL** option. Multiple names require multiple specifications. Same behavior as -vamodel flag, see also "-vamodel <mod_name1> -vamodel <mod_name2> ..." on page 56.

- **SPMODEL**=subckt_name

  Used in conjunction with the **.HDL** command. See ".HDL" on page 678 for further information. The subcircuit name specified with this option has higher priority than a Verilog-A module of the same name when both exist. Only one subcircuit name can be specified on each **SPMODEL** option. Multiple names require multiple specifications. Same behavior as -spmodel flag, see also "-spmodel <subckt_name1> -spmodel <subckt_name2> ..." on page 54.

- **VAOPTS**=options_string

  Used in conjunction with the **.HDL** command. See “.HDL” on page 678 for further information. Specifies options for the Verilog-A compiler. For example:

  ```
  .option vaopts=”-lrm10”
  .verilog file.va
  ```

  The Verilog-A compiler will be called with the -lrm10 option.

Other Options

- **CTEPREC**

  With this option, fundamental physical constants are as follows (SI units):

  Free space permittivity (epso) = $8.854187817 \times 10^{-12}$;
  Boltzmann constant (boltz) =$1.38065812 \times 10^{-23}$;
  Electron charge (charge) = $1.6021773349 \times 10^{-19}$;
  Twice pi (twopi) = $4.0 \times \mathrm{asin}(1.0)$;

  By default, Spice2G6 values are used:

  epso = $8.854214871 \times 10^{-12}$;
  boltz = $1.3806226 \times 10^{-23}$;
  charge = $1.6021918 \times 10^{-19}$;
  twopi = 6.283185308;

- **DCLOG**=VAL

  When Eldo is used with a circuit containing digital gates, a DC analysis is run at first, then the output of the digital gates is updated, and then another DC analysis is run, that is, there are relaxations between DIGITAL and Eldo. There are **DCLOG** relaxations. Default is 1.

- **EPSO**=VAL

  Allows overwriting of the `EPSO` value.

- **MAXNODEORD**

  If number of nodes in the netlist is less than **MAXNODEORD**, then in the **OP** node table, node names are listed by alphabetic order. If the number of nodes exceeds **MAXNODEORD**, node names are listed in the order that Eldo has chosen. Default is 300.

- **NODUPINSTERR**

  When specified in conjunction with the `stver` flag/option, will change `ERROR 838` into `WARNING 204`. This option is ignored if not specified with the `stver` flag/option.

- **NOELDOSWITCH**

  When specified with the `-compat` flag/option, it informs Eldo that devices beginning with an *S* are not switches (Eldo default) but S-parameter block instantiations. This option is ignored if not specified with the `compat` flag/option.

- **NOINIT**

  Prior to an AC or Transient run with UIC active (that is, no DC analysis to be performed), Eldo performs a logical initialization, unless **NOINIT** is defined.

  The concept behind this ‘logical initialization’ is as follows: consider a PMOS element, with 0 V imposed on the gate (via an independent voltage source for instance), and the Source connected to power supply VDD. Then, if Drain has not been initialized, Eldo would initialize its value to VDD, and the value will propagate as far as it can. This algorithm would correctly initialize CMOS designs.

- **NOFNSIEM**

  The Integral Equation Method (IEM) is used in Transient Analysis to solve FNS functions. Specifying **NOFNSIEM** reverts to the implementation based on state variables.

- **SEARCH**=path1[{:path2}]

  When Eldo does not find a subcircuit called `<name>` or a model definition called `<name>`, it will look for the file `path1/<name>.inc`. The `name` is converted into lowercase before searching, and, if more than one **SEARCH** option is given, Eldo will search all the directories in the path specified by the user in the order that they appear inside the input file.

  If more than one path is specified it will search all of the paths. Each path name must be separated by a colon. There is no limit to the number of paths that can be specified.

  This has the same effect as using `-searchpath` in the Eldo instantiation. See “Running Eldo from the Command Line” on page 41.

- **VAMAXEXP**=val

  In Verilog-A, the argument of any exponential is limited to **VAMAXEXP**. Default is 80.0. This is to limit exp(value) when value is too large.

- **ZCHAR**=VAL

  Characteristic impedance of the circuit. Default is 50 Ohms.

Chapter 12
Transient Noise Analysis

Introduction to Transient Noise Analysis

In most analog design applications, knowledge of noise levels generated by the circuit is of great importance. In all traditional SPICE-like simulators, noise computation is only available in AC analysis. In this case the circuit is assumed to have a fixed bias (DC operating point), and noise simulation can only be applied to circuits working under small signal conditions. For this reason, many applications cannot be simulated and their noise performance is unknown until physical measurement is possible on the manufactured design. The only other method available to obtain this important information is to perform tedious hand calculations (when possible).

Eldo provides a solution to this noise analysis problem:

NOISE SIMULATION DURING TRANSIENT ANALYSIS

Transient noise simulation can be applied to all types of circuit without restriction. To perform transient noise analysis, physical noise of electrical devices is emulated by time dependent current sources. The frequency characteristics of these sources are referred to the noise models of the noisy components. The method used is simple, fast and does not disturb the simulated behavior of the circuit because the noise signals introduced are continuous and fully deterministic.

Simulation results from transient noise analysis include:

- Transient response of the circuit, as generated by a pure transient analysis.
- Transient response with the added noise generated by the circuit (plotted as an oscillogram of experimental measurements).
- The RMS value of the noise versus time which gives the accuracy of the ideal transient response.

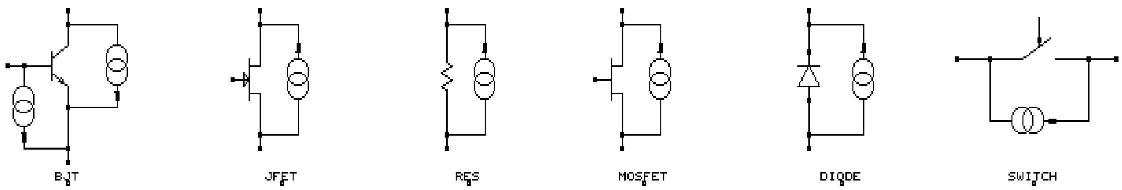
  To compute the RMS value of the generated noise, Eldo performs a normal transient analysis of the circuit, plus ***N*** transient simulations which include noise sources within the noisy devices. The RMS value of noise is computed at each time-step as follows:

$$RMS(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(V_i(t) - V_o(t))^2}$$

Where $V_o(t)$ represents the noiseless transient response of the circuit, and $V_i(t)$ is the i^{th} transient response of the circuit including the noise sources; N is the number of noise simulations performed.

To perform transient noise analysis, Eldo adds a noise contribution to the same components as in AC noise analysis (R, M, B, J, D, see below). Moreover, an additional noise model, connected with the Switch macromodel (S), has been introduced. It is therefore possible to simulate noise in switched capacitor circuits (see “Example 2—Switched Capacitor Filter” on page 1031).

Figure 12-1. Circuit Components with their Added Noise Sources



Please refer to the noise models in the respective sections of the Device Models chapter.

.NOISETRAN Command

```
.NOISETRAN FMIN=VAL FMAX=VAL NBRUN=VAL [OPTIONS]
```

This command is used to control the transient noise analysis of a circuit and must be used in conjunction with a transient analysis (**.TRAN**).

Parameters

- **FMIN**, **FMAX**

  Define the range of the noise frequency band (same function as **FSTART** and **FSTOP** in AC noise analysis).

Note

FMIN and **FMAX** define the frequency band of the noise sources. This frequency range may sometimes not correspond to the noise frequency band at the output of the circuit. For instance, the band (**FMIN**, **FMAX**) does not correspond to the output noise frequency band in the case of filters, oscillators and mixers that exhibit frequency conversion.

- **NBRUN**

  Defines the number of noise simulations to perform (those which include the noise sources). This parameter defines the accuracy of the noise analysis.

Caution

Care must be taken when setting the **NBRUN** parameter as it strongly influences the CPU time used by the simulation.

- OPTIONS

  Other options are available for more advanced and more efficient use of the command.

Please refer to ".NOISETRAN" on page 747 for more details.

Two methods exist to generate transient noise signals. One uses a sum of sine waves with random phases. It is activated when **FMIN** is not zero. The second method, only activated when **FMIN**=0, uses gaussian random variables to generate noise sources. A noise source, with given frequency characteristics, will produce two different noise signals depending on the method used to generate them. These signals will have different instantaneous values but will have the same power and same frequency content.

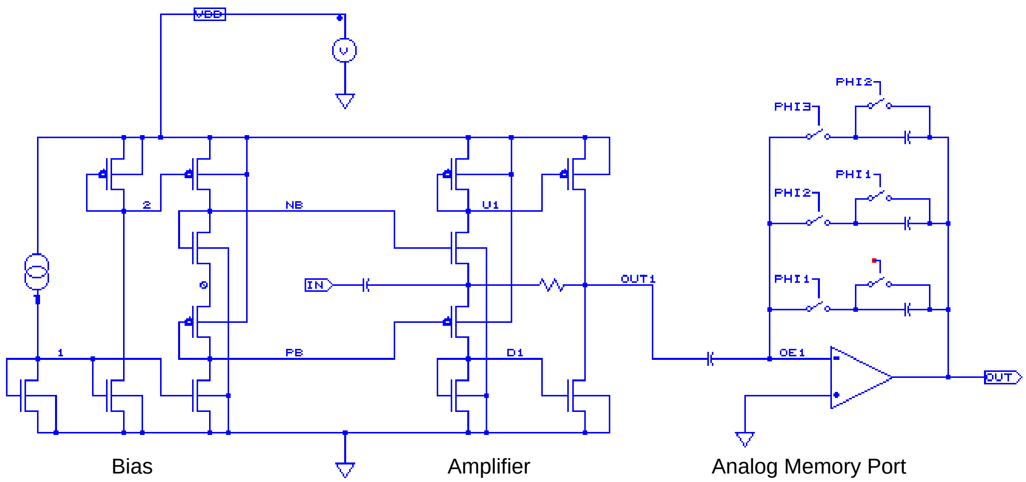
During the transient noise analysis, Eldo generates noise sources in the time domain for each noisy component. These noise sources are generated as a sum of **NBF** sinusoids distributed between **FMIN** and **FMAX**. It means that the generated noise sources have a spectrum of discrete points and the energy of the noise sources is localized in a limited number of frequency points. When a frequency band is wide the corresponding generated noise source may have a poor frequency resolution.

Consequently, an alternative algorithm was developed to generate the noise sources, leading to a continuous spectrum in the frequency band zero to **FMAX**. This algorithm is approximately twice as fast and with more accurate results. However, it cannot generate a noise source with a frequency band beginning at a value of **FMIN** other than zero. This algorithm is automatically activated by specifying **FMIN** as zero. When **FMIN** is different from zero, the method using sinusoids is then used. Both methods can work together, one for some components, the other method for different noisy devices.

The alternative method (when **FMIN**=0) is used to generate flicker noise and white noise sources (thermal and shot noise). It is not possible to have a flicker noise source with a frequency band starting from zero, therefore the generated flicker noise source will have a white spectrum from zero to F1 and a flicker noise spectrum from F1 to **FMAX**. The frequency F1 is calculated from the transient simulation duration: F1=1/TSTOP.

Example 1—High-rate Particle Detector

Figure 12-2. High-rate Particle Detector Circuit



This circuit was designed at CERN and is used in high-rate particle detection. It comprises a front-end pre-amplifier and an analog memory port. Performance analysis of the complete circuit is given in the following sections. Transient simulations have been performed and noise characteristics investigated. The netlist for the circuit, describing the analysis performed, can be found after the simulation results.

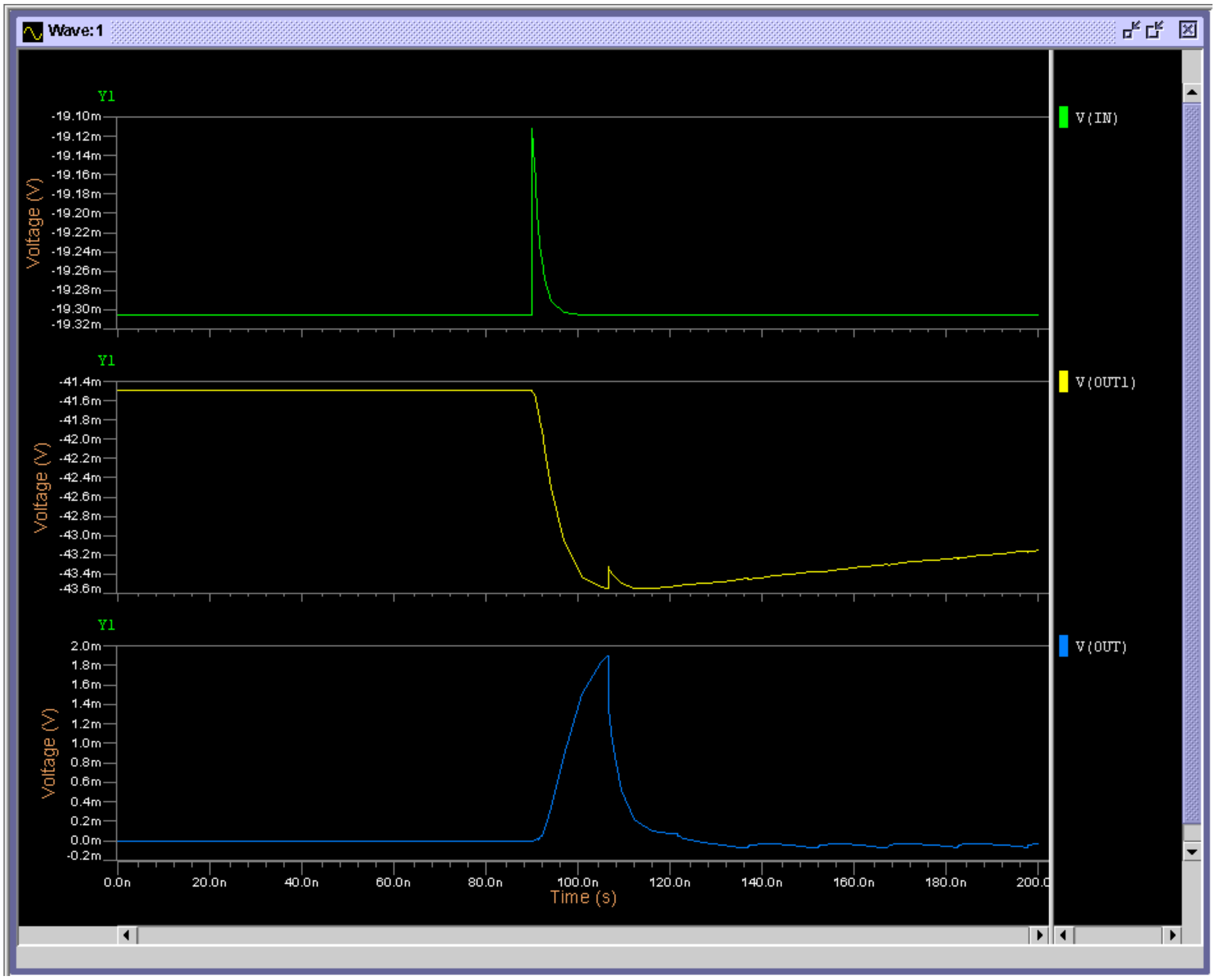
Description of the Particle Detector Behavior

The pre-amplifier has a rise time of less than 20ns and a large fall time constant compared to the rise time. Therefore, for this application it can be considered as an integrator. The function of the analog memory port is signal storage and noise shaping. It takes successive samples of the signal into the different feedback capacitors. The complete system of analog memories acts as a charge sampler and is equivalent, from a signal processing point of view, to a discrete differentiator.

Shown below is the signal at the input, the voltage at the amplifier output and the signal at the output of the complete system. The simulation was performed for an input charge of 2.5 fC across a 5pF input capacitor and with a 15ns clock period.

The clock of the analog memory port is synchronized with the input signal in order to store the maximum amount of charge in the first feedback capacitor. We can note that most of the charge is deposited in one storage capacitor within the 15ns clock period.

Figure 12-3. Simulation Results—Input & Output Signals



Analysis of the Noise Performance

The resolution of this detection system is a function of the noise generated by the circuit. It is therefore the major performance criterion of the device.

The noise performance of the complete circuit strongly depends on the performance of the pre-amplifier. The analog memory port does not generate noise, it only shapes the noise generated by the amplifier.

A number of transient noise simulations were performed on the complete circuit.

Figure 12-4. Noise at Amplifier O/P

Figure 12-4 shows the results of several transient simulation runs including noise sources, with different initial conditions and the RMS value of the noise at the amplifier output calculated from the different runs described above.

Note

The RMS value of the noise increases with time and the Signal to Noise Ratio is limited by the low frequency noise components.

Figure 12-5. Noise at Analog Memory O/P

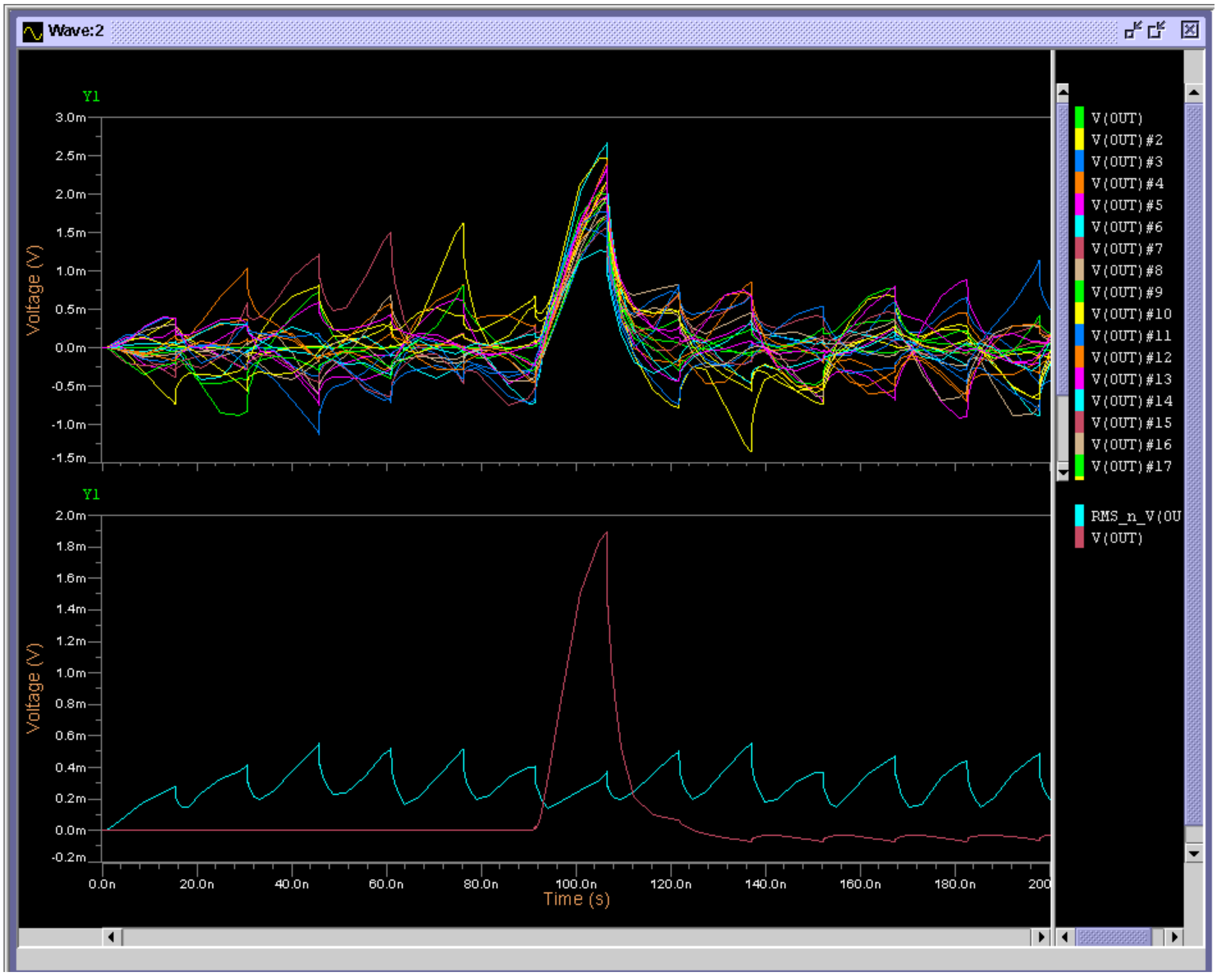


Figure 12-5 shows the curves related to the different runs with the noise sources, the signal at the output of the complete circuit, and the RMS value of the corresponding noise.

Note

The effect of the analog memories on the noise behavior acts as a Correlated Double Sampling, meaning that output noise is reset at each period of the clock.

For this type of circuit, the noise performance is expressed in terms of Equivalent Noise Charge referred on the input (***ENC***):

$$ENC = \frac{Noise}{Signal} Qin$$

Where ***Noise*** is the RMS noise value at the output (extracted from the figure above), ***Signal*** is the signal at the output and ***Qin*** is the charge injected across the input capacitor (in e^-). In this case, the ***ENC*** is about **2000e**$^-$ across Cin (***Noise*** = 0.9mV, ***Signal***=7mV, ***Qin***=2.5 fC). This circuit has been manufactured and experimental measurements match the simulation results very closely.

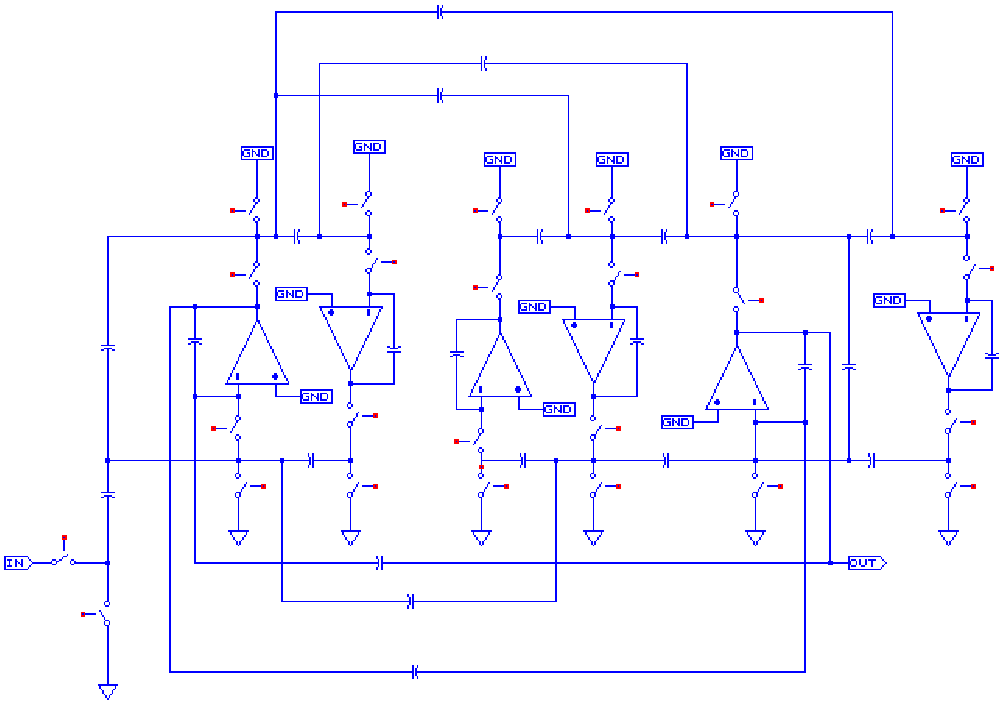
Netlist for Example 1

```
********* MOS MODELS *************
.model nmost nmos level=3 vto=0.75 gamma=0.4 phi=0.457
+ nsub=2.0e15 theta=0.05 ld=0.2e-06 xj=0.3e-6 tox=32.5e-9
+ delta=0.5 uo=700 rsh=37 eta=0.03 cj=20e-5 cjsw=3.5e-10
+ cgso=2.1e-10 cgdo=2.1e-10 mj=0.6 mjsw=0.4 cgbo=8.3e-10
+ pb=0.6 vmax=100e3 kappa=0.48 kf=16.8e-28 nfs=1e10
.model pmost pmos level=3 vto=-0.75 gamma=0.4 phi=0.6
+ nsub=1.0e16 theta=0.14 ld=0.3e-06 xj=0.4e-6 tox=32.5e-9
+ delta=1.5 uo=220 rsh=85 eta=0.08 cj=33e-5 cjsw=3.5e-10
+ cgso=3.1e-10 cgdo=3.1e-10 mj=0.42 mjsw=0.3 cgbo=8.3e-10
+ pb=0.6 vmax=206e3 kappa=15 kf=7.04e-29 nfs=1e10
****** SWITCH MACROMODEL *******
.MODEL swi NSW vh=0.1 vth=2 ron=1 c1=0 c2=0 crec=0
****** CIRCUIT DESCRIPTION *******
**** THE BIAS
Ipol vdd 1 60u
mn1 1 1 vss vss nmost w=35.5u l=5u
mnt1 2 1 vss vss nmost w=35.5u l=5u
mpt1 2 2 vdd vdd pmost w=70u l=5u
mnb2 pb 1 vss vss nmost w=35.5u l=5u
mpb2 nb 2 vdd vdd pmost w=70u l=5u
mnb1 nb nb 0 vss nmost w=540u l=1.5u
mpb1 pb pb 0 vdd pmost w=1080u l=1.5u
**** THE PRE-AMPLIFIER
mni1 u1 nb in vss nmost w=540u l=1.5u AD=+3.375E-09
+ AS=+3.6E-09 PD=+1.635E-03 PS=+1.744E-03
mpi1 d1 pb in vdd pmost w=1080u l=1.5u AD=+3.375E-09
+ AS=+3.6E-09 PD=+1.635E-03 PS=+1.744E-03
mni2 d1 d1 vss vss nmost w=35.5u l=5u AD=+9.0E-11
+ AS=+1.79E-10 PD=+5.0E-05 PS=+9.3E-05
mpi2 u1 u1 vdd vdd pmost w=70u l=5u AD=+9.0E-11
+ AS=+1.79E-10 PD=+5.0E-05 PS=+9.3E-05
mno1 out1 d1 vss vss nmost w=35.5u l=5u AD=+9.0E-11
+ AS=+1.79E-10 PD=+5.0E-05 PS=+9.3E-05
mpo1 out1 u1 vdd vdd pmost w=70u l=5u AD=+9.0E-11
+ AS=+1.79E-10 PD=+5.0E-05 PS=+9.3E-05
Cin inin in 5pf
Rfb in out1 100meg
Cout out1 0 .2pf
Vdd vdd 0 3V
Vss vss 0 -3V
**** ANALOG MEMORY PORT
Coe1 out1 oe1 0.4pF
S1 phi1 oe1 s1 swi
Sr1 phi3 s1 out swi
Cfb1 s1 out 0.4pF ic=0
Vphi1 phi1 0 pulse(-5 5 .1n .1n .1n 15n 45.6n)
S2 phi2 oe1 s2 swi
Sr2 phi1 s2 out swi
Cfb2 s2 out 0.4pF ic=0
Vphi2 phi2 0 pulse(-5 5 15.3n .1n .1n 15n 45.6n)
S3 phi3 oe1 s3 swi
Sr3 phi2 s3 out swi
Cfb3 s3 out 0.4pF ic=0
Vphi3 phi3 0 pulse(-5 5 30.5n .1n .1n 15n 45.6n)
```

```
**OTA definition**
Gamp 0 out 0 oe1 .002
Ramp out 0 0.5Meg
Camp out 0 1pF
Cpip oe1 0 1pF
Ritg out oe1 1g
.ic v(oe1)=0 v(s1)=0 v(s2)=0 v(s3)=0 v(out)=0
**** TRANSIENT NOISE SIMULATION
Vin inin 0 pwl(0 0 90n 0 90.1n 0.3mv 600n 0.3mv)
.tran 200n 200n
.noisetran fmin=100k fmax=100meg nbrun=20
.plot tran v(in)
.plot tran v(out1)
.plot tran v(out)
**** AC ANALYSIS OF THE AMPLIFIER
Vin in 0 dc 0 ac 1
.ac dec 10 100k 100meg
.noise V(out1) Vin 10
.plot ac vdb(out1)
.plot noise inoise onoise
.option thermal_noise=2
.end
```

Example 2—Switched Capacitor Filter

Figure 12-6. Switched Capacitor Filter Circuit Schematic

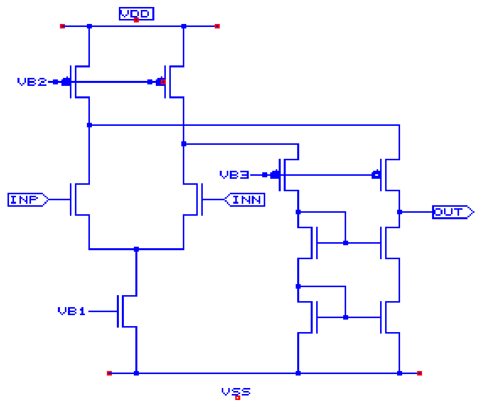


This second example, shown above, is a 6th order switched capacitor bandpass filter used in telecommunications applications. Complete simulation results of the circuit and its noise performance are given in this section. To increase simulation speed, modeling is implemented at a macromodel level rather than transistor level. The Eldo macromodels **SWITCH** and **OPA** are used. We will first describe the macromodels used and simulation results follow thereafter. Finally the netlist describing the simulation conditions is listed after the simulation results.

Characterization of the Amplifier

The structure of the amplifier is a classical folded cascade. AC and transient simulations have been performed at a transistor level in order to determine the macromodel parameters of the circuit shown below:

Figure 12-7. Amplifier Schematic



A subcircuit composed of an `OPA1` macromodel, and a voltage source representing the equivalent noise referred at the input, is used for the simulation of the complete circuit.

Figure 12-8. AC & Noise Simulation Results of Amplifier

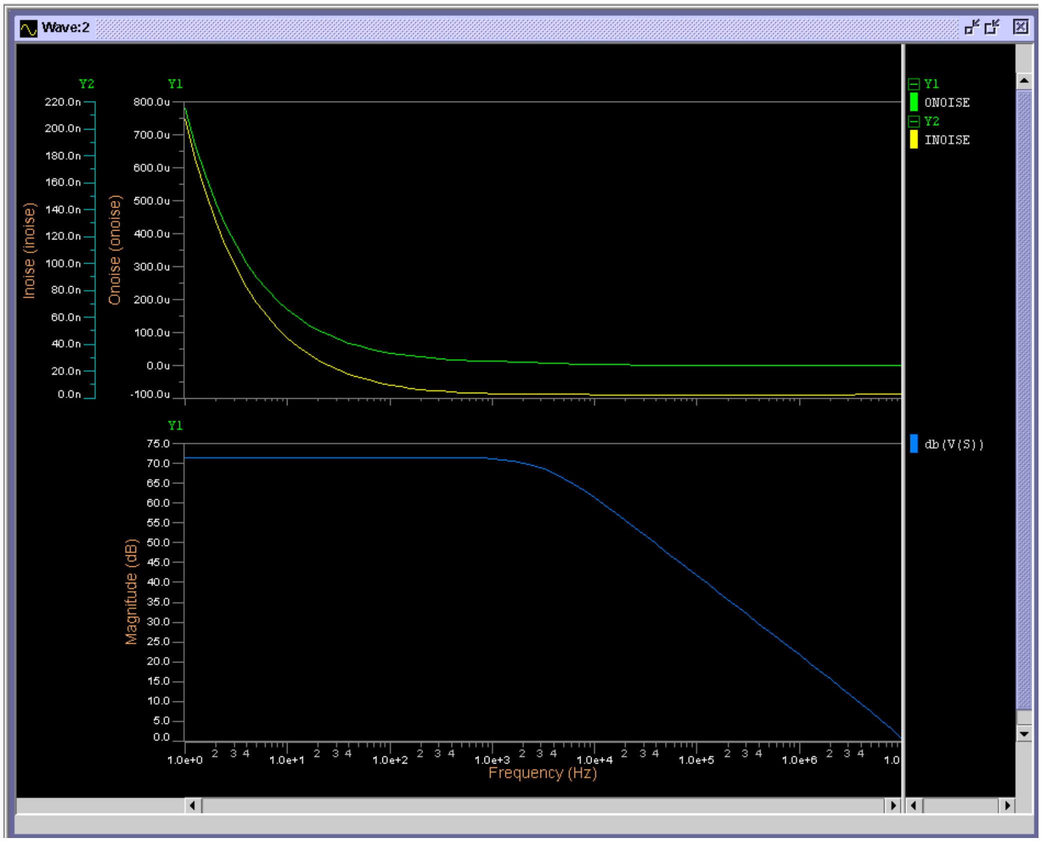
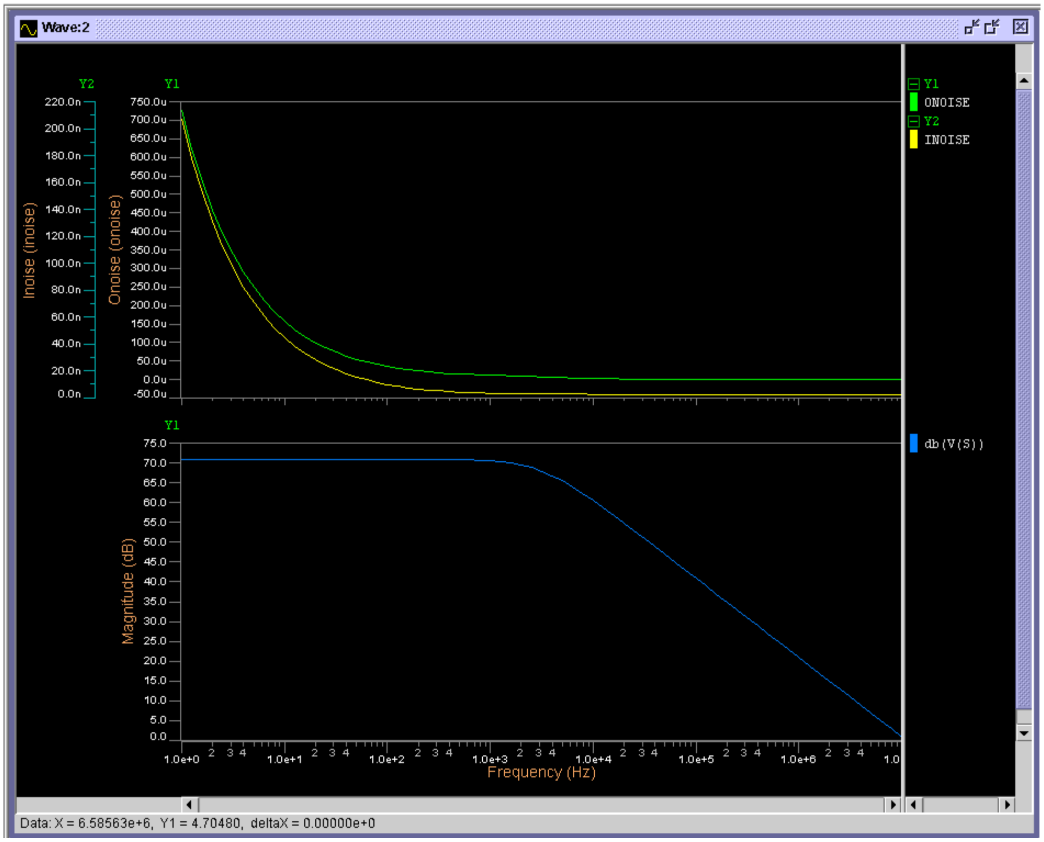


Figure 12-9. AC & Noise Simulation of Amplifier Macromodel



The dominant pole of the amplifier is defined by the output impedance and load capacitance of the amplifier. Comparison of simulation results between the amplifier and its macromodel are shown in Figure 12-8 and Figure 12-9.

Netlist Used for the Amplifier Simulations

AOP Analysis at Transistor Level

```
AOP ANALYSIS
*** AOP Analysis at transistor level
.GLOBAL VDD VSS
.MODEL MN NMOS NIV=6 EOX=200E-10 MU0=520 DPHIF=0.8 DW=1.0E-6
+ DL=0.1E-6 VT0=0.75 KB=0.62 REC=0.1E-6 TG=0.08 VL=1.0E5
+ GL=0.5E-6 KL=0.3E-6 KW=0.2E-6 DINF=0.3 GW=0.4E-6
+ LDIF=3.3E-6 CDIFS0=1.4E-4 CDIFP0=8E-10 VE=20E4 LMIN=1.2E-6
+ WMIN=3.4E-6 RSH=525 AF=1.33 Kf=2.7e-26
.MODEL MP PMOS NIV=6 EOX=200E-10 MU0=190 DPHIF=0.8 DW=1.0E-6
+ DL=0.3E-6 VT0=-0.8 KB=0.36 REC=0.1E-6 TG=0.13 VL=2E5
+ GL=0.34E-6 KL=0.2E-6 KW=0.4E-6 DINF=0.12 GW=0.22E-6
+ LDIF=3.3E-6 CDIFS0=3.2E-4 CDIFP0=8E-10 VE=10E4 LMIN=1.4E-6
+ WMIN=3.4E-6 RSH=1225 AF=0.89 KF=2e-29
```

```
.SUBCKT BIAS VB VB1 VB2 VB3
M1 VSS VB 4 VSS MN L=4.00U W=66.00U
M3 VSS VB VB VSS MN L=4.00U W=66.00U
M5 VB1 VB1 VSS VSS MN L=4.00U W=37.00U
M6 VB1 VB3 VB3 VSS MN L=8.00U W=144.60U
M9 VB3 VB3 8 8 MP L=1.40U W=97.20U
M11 4 4 VB2 VB2 MP L=4.00U W=241.00U
M15 VDD VB2 8 VDD MP L=3.00U W=96.40U
M17 VDD VB2 VB2 VDD MP L=3.00U W=96.40U
C19 VSS VB3 6.32179E00PF
C20 VSS VB2 6.32179E00PF
C21 VB1 VSS 6.32179E00PF
.ENDS
.SUBCKT OPAMP AOUT INP INN VB1 VB2 VB3
M1 VSS 13 11 VSS MN L=10.00U W=194.40U
M5 VSS VB1 12 VSS MN L=4.00U W=118.00U
M7 3 INP 12 VSS MN L=1.20U W=669.00U
M17 4 INN 12 VSS MN L=1.20U W=669.00U
M27 VSS 13 13 VSS MN L=10.00U W=194.40U
M31 13 14 14 VSS MN L=1.50U W=25.60U
M35 AOUT 14 11 VSS MN L=1.50U W=25.60U
M39 3 VB2 VDD VDD MP L=2.00U W=178.00U
M43 4 VB2 VDD VDD MP L=2.00U W=178.00U
M47 3 VB3 14 VDD MP L=1.40U W=194.80U
M51 4 VB3 AOUT VDD MP L=1.40U W=194.80U
.ENDS
***
vvdd vdd 0 2.5
vvss vss 0 -2.5
Xpol vb vb1 vb2 vb3 bias
Xaop s inp inn vb1 vb2 vb3 opamp
Rcha vdd vb 50k
Ccha s 0 50p
vinn inn 0 ac 1
vinp inp 0 0
.ac dec 10 1 10meg
.noise v(s) vinn 10
.plot noise onoise inoise
.plot ac vdb(s)
.option flicker_noise=1
.end
```

AOP Analysis at Macromodel Level

```
********************************************
*** AOP analysis at macromodel level
.MODEL AMP OPA LEVEL=1 VOFF=0 IMAX=200UA
+      CIN=1.0E-12 RS=1Meg
+      GAIN=3500 FNDP=150Meg
+      VSAT=2.5 CMRR=-100db
.subckt opamp o1 o2 i1 i2
opax e1 i2 o1 o2 AMP
Vinoi e1 i1 noise 7.86e-18 4.29e-14 1.33
.ends
```

```
Xaop s 0 0 inn opamp
Ccha s 0 50p
vinn inn 0 ac 1
.ac dec 10 1 10meg
.noise v(s) vinn 10
.plot noise onoise inoise
.plot ac vdb(s)
.end
```

Switch Noise Model

Instead of MOS transistors, a switch macromodel is used to perform the complete circuit simulations. To compute transient noise simulations, noise sources have been added to the switches. We have considered that a switch only generates thermal noise. A current source has been included between the source and the drain and its Power Spectral Density is a function of $\boldsymbol{R_{on}}$, as follows:

$$S_1 = \frac{4kT}{R_{on}}$$

Simulation Results of the Complete Circuit

Transient simulation results are shown below. In Figure 12-10 the first curve represents the impulse response of the circuit. The second curve represents the RMS value of noise generated at the output of the filter in the frequency band (1 Hz, 50 kHz). We can see that this result is about 5 mV. This has been obtained by performing about 20 runs including noise sources. This curve would be smoother if more runs had been performed. Figure 12-11 shows the Fourier transform of the circuit output; it therefore represents the frequency response of the filter.

Figure 12-10. Simulation Results of the Filter

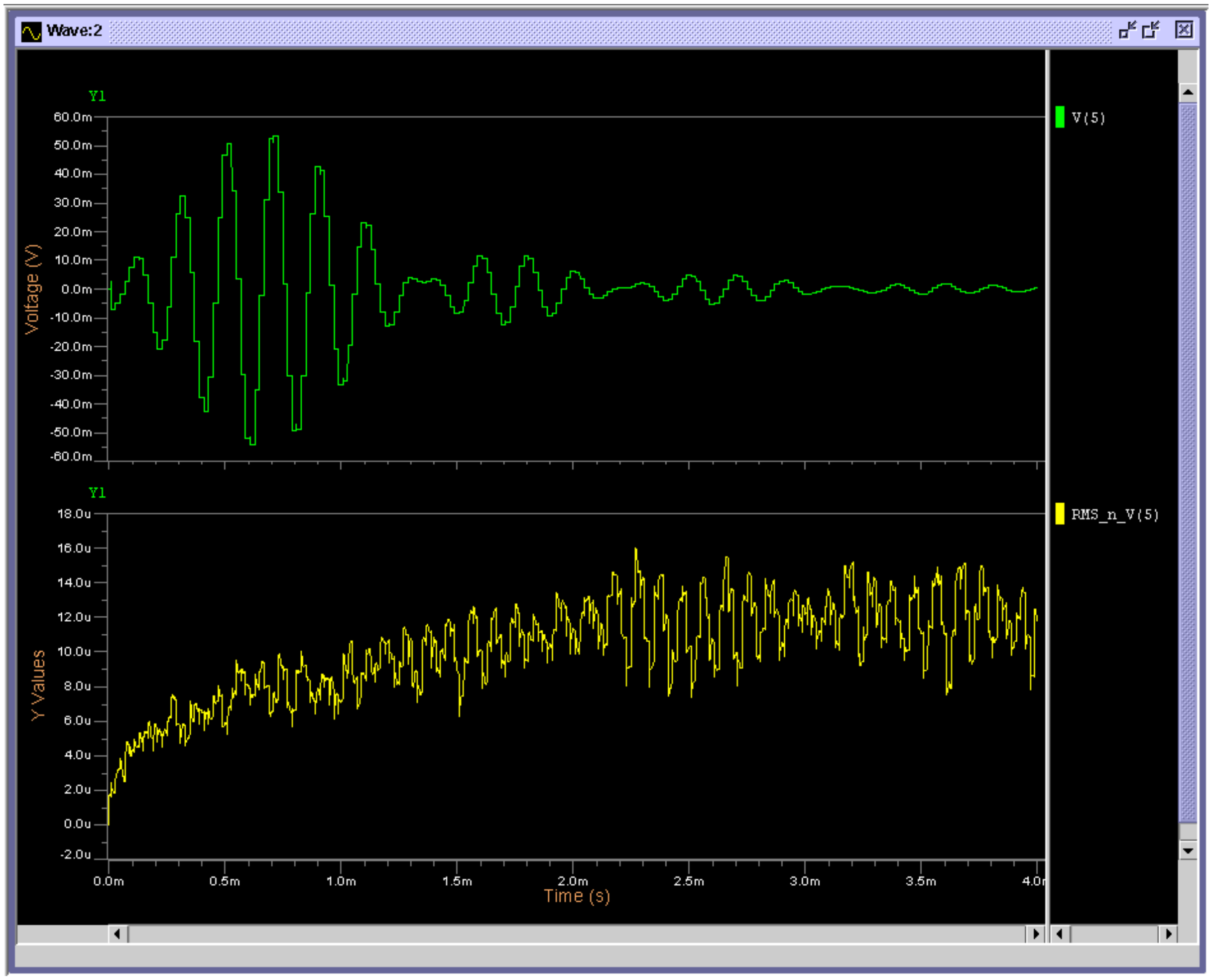
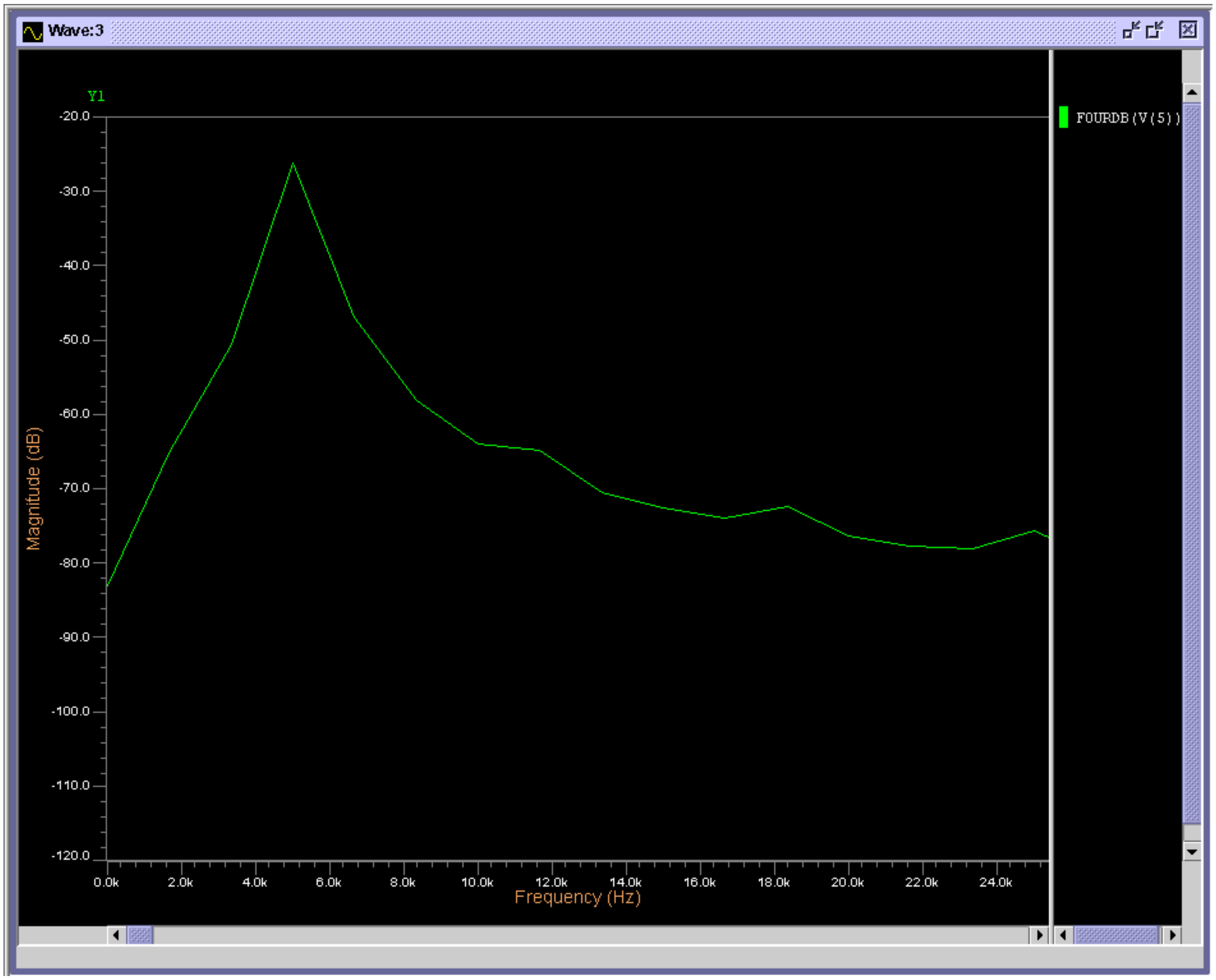


Figure 12-11. Frequency Response of the Filter



Netlist for Example 2

```
* 6TH ORDER BAND-PASS FILTER *
.GLOBAL VDD VSS
VVDD VDD 0 2.5
VVSS VSS 0 -2.5
.MODEL AMP OPA LEVEL=1 VOFF=0 IMAX=200UA CIN=1.0E-12 RS=1MEG
+ GAIN=3000 FNDP=150MEG VSAT=2.5 CMRR=-100DB
.MODEL SWI NSW VH=0.5 VTH=0.4732 GOFF=0.01U RON=1.5K
.SUBCKT OPAMP O1 O2 I1 I2
VINOI E1 I1 NOISE 7.86E-18 4.28E-14 1.33
OPAX E1 I2 O1 O2 AMP
.ENDS
.SUBCKT IT PHI INP OUT
S0 PHI INP OUT SWI
.ENDS IT
```

```
*** INTEGRATOR 1
 XS1 C 9 0 IT
 XS2 CB 9 8 IT
 XS3 CB 10 0 IT
 XS4 C 10 7 IT
 XS5 C 11 0 IT
 XS6 CB 11 1 IT
 XS7 C 12 0 IT
 XS8 CB 12 4 IT
 XS9 C 13 0 IT
 XS10 CB 13 2 IT
*** INTEGRATOR 2
 XS11 CB 15 0 IT
 XS12 C 15 14 IT
 XS13 CB 16 5 IT
 XS14 C 16 0 IT
*** INTEGRATOR 3
 XS15 C 18 0 IT
 XS16 CB 18 17 IT
*** INTEGRATOR 4
 XS17 CB 20 0 IT
 XS18 C 20 19 IT
 XS19 CB 21 3 IT
 XS20 C 21 0 IT
*** INTEGRATOR 5
 XS21 C 23 0 IT
 XS22 CB 23 22 IT
 XS23 C 24 0 IT
 XS24 CB 24 6 IT
*** INTEGRATOR 6
 XS25 CB 26 0 IT
 XS26 C 26 25 IT
*** OUTPUT NODE 5
X1 1 0 0 8 OPAMP
X2 2 0 0 14 OPAMP
X3 3 0 0 17 OPAMP
X4 4 0 0 19 OPAMP
X5 5 0 0 22 OPAMP
X6 6 0 0 25 OPAMP
X7 OUT 0 5 OUT OPAMP
```

```
C20675 10 9 9.23530E-01PF
C20678 9 11 4.52374E-01PF
C20680 12 9 8.43906E-01PF
C20683 13 9 6.42120E00PF
C20698 8 1 9.10828E00PF
C20719 8 5 2.66016E-01PF
C20720 12 18 7.09376E-01PF
C20722 17 3 1.26865E00PF
C20725 12 23 1.64195E00PF
C20729 23 16 1.23994E00PF
C20732 24 23 6.27077E00PF
C20747 22 5 8.70857E00P
C20767 22 1 7.09376E-01PF
C21108 11 15 9.29551E00PF
C21124 16 15 2.66016E-01PF
C21125 2 14 1.70001E01PF
C21153 11 20 9.73922E-01PF
C21155 16 20 7.09376E-01PF
C21157 20 21 6.92543E00PF
C21169 4 19 1.02160E01PF
C21186 16 26 3.50407E00PF
C21192 11 26 2.66016E-01PF
C21193 6 25 6.22576E00PF
*** INPUT SIGNAL AND CLOCK ***
VIN 7 0 pwl (0 0 200n 1 11u 1 11.2u 0)
VCLK00 C 0 PWL (0 -5 200n +5 10000n +5 10200n -5 20000n -5 R)
VCLKB00 CB 0
+ PWL (0 +5 200n -5 10000n -5 10200n +5 20000n +5 R)
.TRAN 1M 4M
.NOISETRAN FMIN=1 FMAX=50K NBRUN=20
.OPTION FREQSMP=150K BE
.PLOT TRAN V(5)
.END
```

Chapter 13
Working with S, Y, Z Parameters

Introduction

A set of commands in Eldo allows you to extract the large signal S parameters (Scattering parameters), the Y parameters (Admittance) or the Z parameters (Impedance) in the frequency domain for a specified circuit. The circuit can have any number of ports.

Eldo enables the simulation of circuits including any number of N-port blocks described by a frequency tabulation of their S (Scattering), Y (Admittance) or Z (Impedance) parameters. It does so by reading the S, Y, Z parameter data from a Touchstone® format file.

Simulation Setup for S, Y, Z Parameter Extraction

Special sources must be added at each port of the circuit to be analyzed. The number of the port and the reference impedance for S parameters must be specified in the Eldo control file as follows:

Source Syntax

```
Vyy NP NN IPORT=VAL [RPORT=VAL] [CPORT=VAL] [LPORT=VAL] [MODE=KEYWORD]
Vyy NP NN IPORT=VAL ZPORT_FILE=string [CPORT=VAL] [LPORT=VAL]
+ [MODE=KEYWORD]
Iyy NP NN IPORT=VAL [RPORT=VAL] [CPORT=VAL] [LPORT=VAL] [MODE=KEYWORD]
Iyy NP NN IPORT=VAL ZPORT_FILE=string [CPORT=VAL] [LPORT=VAL]
+ [MODE=KEYWORD]
```

Parameters

- yy

  Name of the port.

- NP

  Name of the positive node.

- NN

  Name of the negative node.

- **IPORT**

  This is a strictly positive number that is unique and is used as the port number: this number is used for naming the outputs (for instance, `.PLOT AC S(1,2)`). An error message will be issued if two port instances have the same value for **IPORT**, or if an **IPORT** is missing (for example maximum **IPORT** number found in the netlist is 4, and there is no instance with **IPORT** 3).

- **RPORT**

  Value of the Reference Impedance in Ohms. Default value is 50Ω.

- **CPORT**

  Capacitor placed in series (for V source) or parallel (for I source) with **RPORT**. Defaults to 0, in which case it behaves like a zero voltage source (i.e. **CPORT** would have no effect).

- **LPORT**

  Inductor placed in series (for V source) or parallel (for I source) with **RPORT**. Defaults to 0.

- **ZPORT_FILE**

  Specifies the Touchstone file name that contains the port source with a complex impedance. Large signal S parameters can be extracted from a complex port impedance.

- **MODE**=**SINGLE**|**COMMON**|**DIFFERENTIAL**

  Mixed-mode S parameter selection.

  SINGLE specifies the port as single ended, it is dedicated to S parameter extraction. Default.

  COMMON and **DIFFERENTIAL** specify that the port is not single ended. Such ports are split into two linked sources that are either common (same amplitude and same phase) or differential (same amplitude but opposite phases). During S parameter extraction a “non-single ended” port is equally common and differential depending on which display is required. During simulation (DC, AC or TRAN) this port is either common or differential depending on the specified mode keyword.

Note

Port numbers in **V**yy instances should range from 1 to the total number of ports without discontinuity. The simulation parameters **FMIN**, **FMAX**, and Number of frequency points for the analysis are specified with a **.AC** command.

S, Y, Z Parameter Extraction

Once the simulation has been setup (see “Simulation Setup for S, Y, Z Parameter Extraction” on page 1041), S, Y, Z parameters can be extracted during simulation by use of the **.PRINT** and **.PLOT** commands as described in the following subsections.

For S-Parameter Extraction

```
.PLOT   AC  SR(i, j)
.PRINT  AC  SI(i, j)
.PRINT  AC  SM(i, j)
.PLOT   AC  SDB(i, j)
.PRINT  AC  SP(i, j)
```

```
.PRINT AC SGD(i, j)
```

For Mixed Mode S-Parameter Extraction

Mixed mode S parameters can be extracted using the following syntax:

```
S[mn]TYPE(i,j)
```

TYPE can be one of the following:

R Real part
M Magnitude
I Imaginary part
DB Magnitude (dB)
P Phase
GD Group Delay

`mn` specifies the mode of ports `i` and `j` respectively, can be one of the following:

cc common-common
dd differential-differential
dc differential-common
cd common-differential
sc single-common
sd single-differential
cs common-single
ds different-single
ss single-single. Default.

The default mixed mode for S parameter extraction is single-single. If no mixed extension is specified on the output the default will change depending on how ports 1 and 2 are setup. The default rule is shown in Table 13-1.

Table 13-1. Default Rule

Port 1	Port 2	Default	Available quantities
Single	Single	**SS**	**S**[ss](1,1) **S**[ss](1,2) **S**[ss](2,1) **S**[ss](2,2)
Single	Balanced	**SD**	**S**ss(1,1) **S**sd(1,2) **S**sc(1,2) **S**ds(2,1) **S**cs(2,1) **S**dd(2,2) **S**dc(2,2) **S**cd(2,2) **S**cc(2,2)
Balanced	Balanced	**DD**	**S**dd(1,1) **S**dc(1,1) **S**cd(1,1) **S**cc(1,1) **S**dd(1,2) **S**dc(1,2) **S**cd(1,2) **S**cc(1,2) **S**dd(2,1) **S**dc(2,1) **S**cd(2,1) **S**cc(2,1) **S**dd(2,2) **S**dc(2,2) **S**cd(2,2) **S**cc(2,2)

Table 13-1. Default Rule

Port 1	Port 2	Default	Available quantities
Balanced	Single	**DS**	**S**dd(1,1) **S**dc(1,1) **S**cd(1,1) **S**cc(1,1) **S**ds(1,2) **S**cs(1,2) **S**sd(2,1) **S**sc(2,1) **S**ss(2,2)

The default can be set using the **.LIN** command, with syntax:

```
.LIN mixedmode2port=dd|dc|ds|cd|cc|cs|sd|sc|ss
```

Example

```
V1 in1 in2 iport=1 MODE=single
V2 in4 in5 iport=2 MODE=differential
.PLOT AC Sdb(1,1) Sscm(1,2) Ssdp(1,2) Sddr(2,2) Scdi(2,2)
```

For Y-Parameter Extraction

```
.PLOT   AC  YR(i, j)
.PRINT  AC  YI(i, j)
.PRINT  AC  YM(i, j)
.PLOT   AC  YDB(i, j)
.PRINT  AC  YP(i, j)
.PRINT  AC  YGD(i, j)
```

For Z-Parameter Extraction

```
.PLOT   AC  ZR(i, j)
.PRINT  AC  ZI(i, j)
.PLOT   AC  ZM(i, j)
.PLOT   AC  ZDB(i, j)
.PRINT  AC  ZP(i, j)
.PRINT  AC  ZGD(i, j)
```

Where **S**xx(i, j), (**Y**xx(i, j), **Z**xx(i, j)) give the influence of port j on port i.

Matrix Parameter Extraction

Once the simulation has been setup (see “Simulation Setup for S, Y, Z Parameter Extraction” on page 1041) then G, H, T, A parameters can be extracted during an AC or FSST simulation by use of the **.PRINT** and **.PLOT** commands as described in the following subsections.

For G-Parameter Extraction

```
.PLOT   AC  GR(i, j)
.PRINT  AC  GI(i, j)
.PRINT  AC  GM(i, j)
```

```
.PLOT   AC  GDB(i ,j)
.PRINT  AC  GP(i, j)
.PRINT  AC  GGD(i, j)
```

For H-Parameter Extraction

```
.PLOT   AC  HR(i, j)
.PRINT  AC  HI(i, j)
.PRINT  AC  HM(i, j)
.PLOT   AC  HDB(i ,j)
.PRINT  AC  HP(i, j)
.PRINT  AC  HGD(i, j)
```

For T-Parameter Extraction

```
.PLOT   AC  TR(i, j)
.PRINT  AC  TI(i, j)
.PRINT  AC  TM(i, j)
.PLOT   AC  TDB(i ,j)
.PRINT  AC  TP(i, j)
.PRINT  AC  TGD(i, j)
```

For A-Parameter Extraction

```
.PLOT   AC  AR(i, j)
.PRINT  AC  AI(i, j)
.PRINT  AC  AM(i, j)
.PLOT   AC  ADB(i ,j)
.PRINT  AC  AP(i, j)
.PRINT  AC  AGD(i, j)
```

Where `Sxx(i, j)`, (`Yxx(i, j)`, `Zxx(i, j)`) give the influence of Port `j` on Port `i`.

Output File Specification

```
.ffile S|Y|Z|G|H|T|A [SINGLELINE] FILENAME [HZ|KHZ|MHZ|GHZ] [RI|MA|DB]
```

Parameters

S Specifies S (Scattering) frequency parameters tabulation.

Y Specifies Y (Admittance) frequency parameters tabulation.

Z Specifies Z (Impedance) frequency parameters tabulation.

G Specifies G (Hybrid-G) matrix parameters tabulation.

H Specifies H (Hybrid-H) matrix parameters tabulation.

T Specifies T (transfer scattering) matrix parameters tabulation.

A Specifies A (chain or ABCD) matrix parameters tabulation.

`FILENAME` Name of the file where the S, Y, Z, G, H, T and A parameters will be stored.

`SINGLELINE` This enables you to obtain the S-parameter file in single line format as shown below:

```
Freq S11 S21 S12 S22
```

`HZ` Specifies the units to be Hz. This is the default.

`KHZ` Specifies the units to be kHz.

`MHZ` Specifies the units to be MHz.

`GHZ` Specifies the units to be GHz.

`RI` Specifies Real Imaginary storage format.

`MA` Specifies Magnitude Angle storage format. This is the default.

`DB` **`MA`** with magnitude in dB.

Two-port noise parameters NFMIN_MAG, GAMMA_OPT_MAG, PHI_OPT and RNEQ are automatically written to the specified output file when a **`.NOISE`** command is specified in the netlist and the circuit to be analyzed is a two-port circuit.

Examples

```
r1 1 2 100k
c1 2 0 10pf
V1 1 0 iport=1 rport=100
V2 2 0 iport=2 rport=20
.ac dec 10 1 100meg
.plot ac sdb(2,1)
.Ffile S sb1.par khz ri
```

In this example, the S parameters of an RC circuit are extracted between 1 Hz and 100 MHz with 10 points per decade. The reference impedance is 100 for `port1` and 20 for `port2`. The magnitude of `S21` is plotted in dB, and the extracted S parameters are stored in the file *sb1.par* with the frequency in kHz. The data is stored in the form of the Real and Imaginary parts.

```
V1 1 0 iport=1 rport=50

R1 1 n1 1k
C1 n1 0 100p
R2 n1 2 1k
Rc1 n1 0 100k

V2 2 0 iport=2 rport=50

.ac lin 21 1meg 21meg
.noise v(n1) V1 3

.plot noise rneq gopt bopt nfmin_mag
.ffile Z Z.par kHz ma
```

In this example the Z parameters are being extracted between 1MHz and 21MHz with 21 analysis points. The extracted Z parameters are stored in the file *Z.par* with the frequency in kHz. The data is stored in the form of the Magnitude Angle. As the circuit is a two-port circuit and there is a **.NOISE** command specified in the netlist then the two-port noise parameters are also stored in the output file. The output file is shown below:

```
! Data from test
# KHZ Z DB R 5.000000E+01
!
1.0000000000000000E+03 6.5542511585029786E+01 -5.7202544106307606E+01
6.4035302689753806E+01 -8.9088186330215692E+01 6.4035302689753806E+01
-8.9088186330215706E+01 6.5542511585029786E+01 -5.7202544106307606E+01
...

2.1000000000000000E+04 6.0025369785355387E+01 -4.3338006361865595E+00
3.7592014242230192E+01 -8.9956576643609139E+01 3.7592014242230199E+01
-8.9956576643609139E+01 6.0025369785355402E+01 -4.3338006361865711E+00

! Noise Data: Nfmin(dB) GammaOpt PhiOpt Rneq/R0
1.0000000000000000E+03 5.2661113010469025E+00 9.6890047604421903E-01
1.7851510536508835E+02 4.8497683522933400E+01
...

2.1000000000000000E+04 2.8441528902447214E+01 9.0554147314402922E-01
1.7956971588917614E+02 3.5225984336136335E+03
```

In the following example the S parameters of the block TWO_PORT_SUBCKT are being extracted between 10kHz and 10MHz with 10 analysis points per decade. The Touchstone file *Z1.par* defines the complex impedance of the source at port 1. The extracted S parameters are stored in the file *subckt.par* with the frequency in kHz. The data is stored in the form of the Magnitude Angle.

```
Vin 1 0 iport=1 zport_file="Z1.par"
Xsub1 1 0 2 0 TWO_PORT_SUBCKT
Vout 3 0 iport=2 rport=50
.ac dec 10 10000 10meg
.plot sdb(1,2) sdb(1,1) sdb(2,2) sdb(2,1)
.ffile S subckt.par KHZ MA
```

Transient Simulation of Circuits Characterized in the Frequency Domain

Introduction

Traditionally, high frequency circuits are characterized and simulated in the frequency domain. This is because of the difficulty of handling extremely short rise times of the order of pico-seconds and the simplicity of frequency measurement.

Today with the technological advent in high frequency circuits, there is a vital need to simulate circuits in the time domain using electrical simulators. This is needed to simulate, for example, a linear (lossy and maybe coupled) interconnection having a non-linear termination. Such problems may be solved in the frequency domain using harmonic balance. If we are interested in using pulse stimuli, the circuit must be analyzed in the time domain. Another example is the microwave simulation of passive elements, either discrete or integrated. A user defined passive element may be simulated by extracting its scattering (S) parameters using a standard Electro-Magnetic (EM) solver and then using this file as an input to an S-Model. This procedure enables the user to simulate this passive element in Eldo either with or without other linear or non-linear elements.

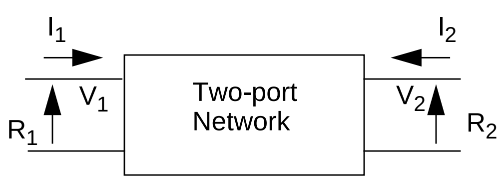
The main object of the S-Model GenLib library is to allow the transient analysis of pre-characterized (in the frequency domain) linear high frequency circuits with any other non-linear components. The circuit is usually characterized by its scattering parameters. S-Model also allows the simulation of circuits characterized using their admittance (Y) or impedance (Z) parameters. The pre-characterized circuit may have any number of ports. The following paragraph gives a brief technical presentation of matrix representation of linear circuits.

Technical Background

A two port network is completely presented by its Z, Y, h or S matrix. As an example, consider the impedance Z matrix:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}$$

It is clear that this matrix relates the currents and voltages at the terminals of a given block:



An equivalent presentation is:

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

In this case, we use the S matrix or scattering parameters. $\boldsymbol{a_1}$ and $\boldsymbol{b_1}$ present the normalized incident and reflected waves at port 1. $\boldsymbol{a_2}$ and $\boldsymbol{b_2}$ present the corresponding waves at port 2. There are direct relationships between $\boldsymbol{a_1}$, $\boldsymbol{b_1}$, $\boldsymbol{a_2}$, $\boldsymbol{b_2}$ and the corresponding $\boldsymbol{I_1}$, $\boldsymbol{V_1}$, $\boldsymbol{I_2}$, $\boldsymbol{V_2}$.

In the case of a two port network, the I and V as a function of a and b is given by the following:

$$V_1 = (a_1 + b_1)\sqrt{R_1}$$
$$I_1 = (a_1 - b_1)/\sqrt{R_1}$$
$$V_2 = (a_2 + b_2)\sqrt{R_2}$$
$$I_2 = (a_2 - b_2)/\sqrt{R_2}$$

For the same case, a and b as a function of I and V is given by the following:

$$a_1 = \frac{V_1 + R_1 I_1}{2\sqrt{R_1}}$$
$$b_1 = \frac{V_1 - R_1 I_1}{2\sqrt{R_1}}$$

$$a_2 = \frac{V_2 + R_2 I_2}{2\sqrt{R_2}}$$
$$b_2 = \frac{V_2 - R_2 I_2}{2\sqrt{R_2}}$$

where $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ are the reference impedances of ports 1 and 2 respectively.

The S-parameters are widely used to characterize high frequency circuits, mainly because they present no difficulty in measurements while the other parameters are difficult to measure. The scattering parameters are not unique; they are defined for a given reference impedance for each port. The reference impedance $\boldsymbol{R}_0$ is usually 50Ω for all ports to facilitate measurement (standard coaxial cable has 50Ω characteristic impedance).

In general, an n-port circuit has an $n \times n$ scattering matrix of the following form:

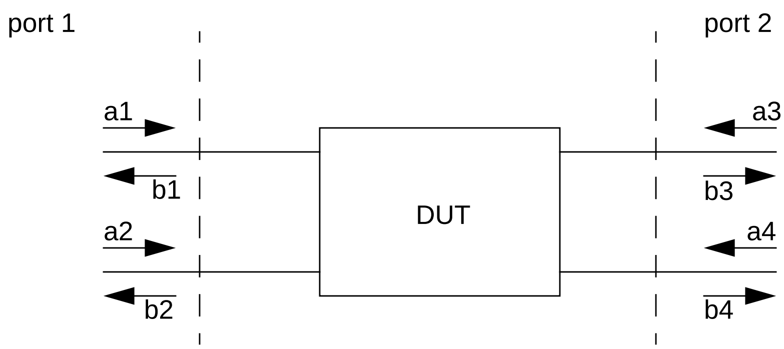
$$\begin{bmatrix} b_1 \\ .. \\ .. \\ b_n \end{bmatrix} = \begin{bmatrix} S_{11} & .. & .. & S_{1n} \\ .. & .. & .. & .. \\ .. & .. & .. & .. \\ S_{n1} & .. & .. & S_{nn} \end{bmatrix} \begin{bmatrix} a_1 \\ .. \\ .. \\ a_n \end{bmatrix}$$

Mixed-Mode S Parameters

Bockelman and Eidenstadt[1] developed a theory for combined differential and common normalized power waves (in terms of even and odd mode). Then it is now possible to characterize multiport networks at high frequencies, especially such device which are simulated by common-mode or differential-mode source, by using the extended S parameter definition. This adaptation, called “mixed-mode S parameter”, addresses differential and common-mode operation, as well as the conversion between the two modes operation.

1. David E. Bockelman William R. Eisenstadt, “Combined Differential and Common-Mode Scattering Parameters: Theory and Simulation” July 1995.

According with this new definition, we can see that a two port S parameters form a 4x4 matrix containing the mixed-mode S parameters (differential-mode, common mode and cross-mode S parameters). Consider the following differential circuit, each port can support the propagation of differential-mode and common-mode waves



The response of this differential circuits to a stimulus can be expressed with the mixed-mode S parameter matrix:

$$\begin{bmatrix} b_{d1} \\ b_{d2} \\ b_{c1} \\ b_{c2} \end{bmatrix} = \begin{bmatrix} S_{dd11} & S_{dd12} & S_{dc11} & S_{dc12} \\ S_{dd21} & S_{dd22} & S_{dc21} & S_{dc22} \\ S_{cd11} & S_{cd12} & S_{cc11} & S_{cc12} \\ S_{cd21} & S_{cd22} & S_{cc21} & S_{cc22} \end{bmatrix} \begin{bmatrix} a_{d1} \\ a_{d2} \\ a_{c1} \\ a_{c2} \end{bmatrix}$$

where the partition labeled S_{dd} are the differential-mode S parameters, S_{cc} are the common-mode S parameter, and S_{cd} and S_{dc} the cross-mode S parameters. The a_{di} and b_{di} are the normalized differential-mode stimulus and response waves; a_{ci} and b_{ci} are the normalized common mode stimulus and response waves. The definition of these normalized waves are:

$$a_{d1} = \frac{1}{\sqrt{2}}(a_1 - a_2)$$

$$a_{c1} = \frac{1}{\sqrt{2}}(a_1 + a_2)$$

$$b_{d1} = \frac{1}{\sqrt{2}}(b_1 - b_2)$$

$$b_{c1} = \frac{1}{\sqrt{2}}(b_1 + b_2)$$

$$a_{d2} = \frac{1}{\sqrt{2}}(a_3 - a_4)$$

$$a_{c2} = \frac{1}{\sqrt{2}}(a_3 + a_4)$$

$$b_{d2} = \frac{1}{\sqrt{2}}(b_3 - b_4)$$

$$b_{c2} = \frac{1}{\sqrt{2}}(b_3 + b_4)$$

In general, an n-port has a 2n x 2n mixed-mode S parameter matrix of the following form:

$$\begin{bmatrix} b_{d1} \\ \dots \\ b_{dn} \\ b_{c1} \\ \dots \\ b_{cn} \end{bmatrix} = \begin{bmatrix} S_{dd11} & \dots & S_{dd1n} & S_{dc11} & \dots & S_{dc1n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ S_{ddn1} & \dots & S_{ddnn} & S_{dcn1} & \dots & S_{dcnn} \\ S_{cd11} & \dots & S_{cd1n} & S_{cc11} & \dots & S_{cc1n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ S_{cdn1} & \dots & S_{dcnn} & S_{ccn1} & \dots & S_{ccnn} \end{bmatrix} \begin{bmatrix} a_{d1} \\ \dots \\ a_{dn} \\ a_{c1} \\ \dots \\ a_{cn} \end{bmatrix}$$

Implementation Issues

Eldo uses three methods to simulate an S-Model given by S, Y, or Z parameters. These methods are briefly described below:

- Complex Pole Fitting (CPF) technique is a method based on complex-pole fitting of the original dependence. During an initial “fitting” stage, the model’s given dependence is represented as a sum of simple first-order components, each one defined by its complex pole and residue. The result of fitting is re-usable; once generated, the list of poles and residues is stored in a **.pls* file and can be used repeatedly for simulations without the need to re-fit. This file has the same name and location as the original data (Touchstone) file.

  The representation of frequency dependence created by fitting allows fast and accurate transient simulation of the S-Model. Simulation progresses linearly in time, using an effective, recursive, convolution-based algorithm. Although poles/residues can be used to build an equivalent circuit, this is not needed or recommended, for performance reasons: the model-evaluation time in CPF is less by a factor of 5-7 than that for the equivalent circuit built from the same poles.

  Due to the very nature of fitting, the CPF method always results in a causal solution. It also has a delay-extraction capability useful when simulating transmission lines or connectors.

- Digital Signal Processing (DSP) technique is an alternative approach that transforms the frequency-domain data into time domain parameters via inverse FFT, Hilbert transform and convolution. Important modifications were made to these basic algorithms to allow both periodic and non-periodic dependencies, to ensure the causality of the system, to account for singularities in matrix representation and to ensure high-speed convolution.

If the circuit frequency response is naturally periodic (as for delay-containing operators) and is given only in a fraction of a single period, the DSP method is recommended to ensure accurate simulation. In this case, the last point given in the input data file should correspond to the half-period of the dependence.

- System identification (SI) technique is a third method that represents S-parameters in the form of a rational function in ‘s’ (Laplace variable). These functions are then converted into systems of linear differential equations so that Eldo can solve them during the transient analysis.

With the above three methods, Eldo can efficiently solve a wide variety of problems. Frequency dependencies can be either quite smooth or with a large number of sharp resonant peaks (up to many hundred). The user may specify input data either with equidistant frequency points, starting from zero or not; or give them in any other way, (for example, logarithmically spaced) relevant to the method of data acquisition.

Of course, one can expect accurate simulations only if the original data is complete and accurate. The frequency dependence given should completely encompass the range of interest, from the lowest to the highest operation frequency. For example, high accuracy at DC is unlikely if the data starts from non-zero frequency. Similarly, accurate simulation of short transitions, lasting for hundred ps, is impossible if the highest point is far below tens of GHz. Also, the data points should be given with good resolution, sufficient to reproduce the shape of the dependencies. For example, many more points are needed to describe a dependence with many sharp peaks than for a smooth one, even if they are both defined in the same frequency range. Finally, the input data should be causal, so that the real and imaginary parts of the frequency dependence satisfy the dispersion Kronig-Kramers relation. In reality, data becomes slightly non-causal due to unavoidable measurement/simulation errors, especially (typically) at higher frequencies. However, serious measurement errors (like taking the negated phase) cause catastrophic non-causality that will lead to improper simulation results.

Applications

S-parameters can originate from the following:

- Simulation of passive networks using an electromagnetic solver.
- Measurements from a passive network (interconnects, transmission lines, passive filters, and so on).
- Frequency-domain simulation of passive networks (AC analysis).
- Data sheets of active or passive devices.

The major applications of S-Model are as follows:

- Transient simulation of microwave linear circuits originally described by its response in frequency domain.
- Transient simulation of interconnects together with non-linear drivers/loads, either for VLSI or GaAs integrated circuits, using either their calculated or measured scattering parameters.
- Lossy transmission-line transient simulation in the presence of either linear or non-linear drivers/loads.
- Transient simulation of arbitrary (or user-defined) passive microwave circuits after extracting their S-parameters (using a standard electromagnetic solver).

Functionality

Basic Functionality

The S-model implemented in Eldo is a building block that makes possible DC, AC, and Transient simulation of circuits with any number of N-ports described by their S (Scattering), Y (Admittance), or Z (Impedance) parameters, in the form of tabulated data in the frequency domain.

The tabulated data is contained in an ASCII data file in the Touchstone® format. This format is briefly described in "Touchstone Data Format" on page 1058. When giving the instance of the model in the Eldo netlist file, you can specify the data file either by setting an index associated with the file's name, or by explicitly defining the name of the file. The optimal of the three algorithms discussed above, Complex Pole Fitting (CPF), Digital Signal Processing (DSP), and System Identification (SI), is selected by the internal Eldo monitor that allows great flexibility of simulation. You can also specify this method directly.

Detailed Functionality

- Simulation of N-ports with no restriction on N. N is determined automatically by the Eldo parser according to the number of pins.
- Any number of instances of the same model, any number of different models.
- All SST and MODSST simulation types are supported.
- DC Simulation.
- AC Simulation. When the CPF algorithm is chosen either by the monitor or the user, the frequency response of the system is computed based on the found poles/residues, to enable smooth and causal simulation results. In the case of the DSP method, AC simulation is performed by interpolation and extrapolation of the tabulated parameters, with the response made symmetrical around the maximal tabulated frequency point, and the obtained spectrum is made periodic. With the SI algorithm, AC simulation produces the frequency response of the transfer function that best fits the tabulated frequency

points. If AC simulation is performed without transient simulation and without any method forced by the user, the response is obtained through interpolation of the given frequency-domain data points.

- Transient Simulation. Transient simulation is obtained through the application of the most appropriate of the CPF, DSP, or SI algorithms.
- Input data can be specified as S, Y, Z, G, H, T or A parameters.
- Tabulated data may have linear, logarithmic, or irregular distribution in its frequency range. Frequency values may start from 0 Hz or any positive value. Any number of points is allowed.
- The choice of the CPF, DSP, or SI algorithm can be forced through a parameter of the model. All algorithms allow any kind of spacing. However, since internally DSP requires linearly spaced K^2+1 data points starting at zero frequency, it uses interpolation and extrapolation of the input data to satisfy this requirement.
- Simulation is possible with the Eldo default options.
- For an N-port block (N>1), port impedances can be identical, or different for each port.
- Speed-optimized C-FAS model.

Instantiating a Block Defined by S-Parameters

```
.MODEL FBLOCK MACRO LANG=C
YNAME FBLOCK PARAM:
+ [M=VAL]
+ [IDX_M=VAL]
+ [NO_DELAY=VAL]
+ [GROUPFIT=VAL]
+ [SYMMETRY=VAL]
+ [FORCE_PASSIVITY=VAL]
+ [FORCE_REFIT=VAL]
+ [EXTRAP_TO_DC=VAL]
+ [POLE_REDUCTION=VAL]
+ [HIGH_PRECISION=VAL]
+ [MAXROW=VAL]
+ [MAXCOL=VAL]
+ [IDX_F=VAL]
+ STRING: FILENAME
+ PIN: IP1 IN1 ... IPN INN
```

The first line (**.MODEL**) is the reference to the C-FAS model, where the entity is called **FBLOCK**. The other lines are the model parameters.

Note

This **.MODEL** line is optional. This declaration is not a requirement of Eldo when referencing embedded Eldo FBLOCK models.

The keyword **PARAM:** precedes the list of parameters, (each one shown in brackets as they are all optional). Most of the options, except for **M**, **IDX_M** and **NO_DELAY**, are specific to the CPF method.

M is a device multiplier parameter, simulating the effect of multiple S-block elements in parallel. Default value is 1.

IDX_M is a parameter that forces a specific algorithm to be used (**IDX_M**=0 forces the CPF algorithm, **IDX_M**=1 specifies DSP, **IDX_M**=2 specifies SI). The default value of **IDX_M** is 0 (CPF).

NO_DELAY is used to allow or prevent delay extraction in the CPF or DSP methods. **NO_DELAY**=0 allows delay extraction, **NO_DELAY**=1 forbids it. The default value is 0.

GROUPFIT=1 is used in CPF to force group fitting instead of individual for every matrix component. As a rule, with this option, fitting requires less effort but this might compromise accuracy. By default, its value is 0 that corresponds to individual fitting.

SYMMETRY=0 disables the default assumption (**SYMMETRY**=1) made in CPF on the fitting stage that the original S (or Y or Z) matrix is symmetric. Matrix symmetry is a valid assumption as long as the S-model describes a reciprocal subcircuit. We cannot simply rely on symmetry of the matrices in the input data. Very often, the input matrices generated by field-solvers or measured from reciprocal systems, are not strictly symmetric, however they should be handled as symmetric.

FORCE_PASSIVITY=`val` enables or disables each of the two different types of passivity enforcement available in the CPF method. These types are (1) pre-fit passivity enforcement, in which the original sampled data is worked with to make it “passive,” and (2) post-fit enforcement, in which poles/residues are corrected in such a way as to make the approximation strictly passive.

FORCE_PASSIVITY=`0` (default) means there is no passivity enforcement.
FORCE_PASSIVITY=`1` activates pre-fit passivity enforcement.
FORCE_PASSIVITY=`2` activates post-fit enforcement.
FORCE_PASSIVITY=`3` activates them both.

Pre-fit passivity enforcement is recommended for all passive devices. It removes occasional passivity violations from the input data (which may result from measurement errors). However, even for the passive data created by pre-fit passivity enforcement, fitting may still result in a non-passive model if this data is defined within a limited frequency range (typical case). With two different methods of passivity enforcement, you can determine the true reason for non-passivity: poor accuracy of the input data or fitting errors. The reason for both could be incomplete frequency range, non-causality, or insufficient resolution of the input data. For causal, accurate, and smooth input data, fitting accuracy is quite high.

If non-reciprocal linear active devices (such as amplifiers or filters) are to be simulated, both **SYMMETRY** and **FORCE_PASSIVITY** should be disabled.

FORCE_REFIT=1 forces fitting in CPF regardless to whether the corresponding *.pls* file is already present or not. This might be needed if we want to redo fitting with different options, such as **FORCE_PASSIVITY**. However, you should be careful in using such an option, it should be disabled after the desired fit is built. By default, **FORCE_REFIT** is disabled.

EXTRAP_TO_DC=1 restores a missing point at zero frequency (DC) by extrapolating the curve from low frequency points given in the Touchstone file. If the DC point is present in the input data, this option has no effect. Compared to the default case (**EXTRAP_TO_DC**=0) it allows, as a rule, to achieve better accuracy in DC simulation when the point at zero frequency is not given.

POLE_REDUCTION=1 (default value) enables the mode of transient simulation in which some of the fitted poles (that are too fast, too slow or too small) are removed in order to speed-up the solution. This mode typically gives up to 30-50% reduction in solution time when the step of the transient solution is fixed. The decision about pole reduction is made from considering the solution step (pole is too "fast"), or the duration of the simulation interval (pole is too "slow"). Therefore, the set of actually used poles is defined "dynamically" from considering the parameters of the **.TRAN** command. The generated list of poles/residues (**.pls*) file remains unaffected. Pole reduction does not considerably affect the solution accuracy. However, if the precise simulation is needed, the option can be disabled by setting **POLE_REDUCTION**=0.

HIGH_PRECISION=1 increases fitting accuracy by allowing more poles than in regular mode (with default value 0). This option can be useful for verification purposes, for example if a "reference solution" is required. However, it is not recommended if the input data itself is not very accurate. Also, since high-precision fitting produces more poles, it makes simulation slower.

MAXROW=VAL sets the limit (**val/2**) to the frequency points of the original dependence used in fitting. By default, **MAXROW=40000**, that corresponds to 20,000 points.

MAXCOL=VAL sets the limit (**val/2**) to the maximal order of complexity for fitting in CPF. By default, **MAXCOL=1500**, that corresponds to a order of complexity of 750. For very complicated (sharp, irregular) dependencies it is sometimes reasonable to reduce the order of complexity, especially if we have reasons not to entirely trust the input data at higher frequencies. As a rule, reducing order of complexity is a better strategy than reducing the number of points to consider (**MAXROW**).

The keyword **STRING:** is to define the name of the touchstone file, containing the input data. Path definition is allowed. Another way of defining the data file is using the parameter **IDX_F**. Note that the parameter **IDX_F** should be defined under the keyword **PARAM:** together with all other parameters, not under the **STRING:** keyword. This parameter defines the index (integer number) VAL associated with the S parameter file (**IDX_F**=VAL implies that the input parameter file is named *sb*VAL*.par*).

The 2×N pins of the N-port model will be connected to nodes `IP1 IN1 ... IPN INN`.

A single reference node is supported. When the number of pins of the **FBLOCK** model is even, Eldo considers that each port has two pins. When the number of pins is odd, Eldo considers the reference pin is the same for all ports (and it is the last pin).

Any model may be instantiated as many times as required with the same or different input data file.

Any **FBLOCK** instance will contribute to the global noise results of **.NOISE** and **.SSTNOISE**. If the Touchstone format file contains noise parameters then they will be used to compute the noise contribution, otherwise the simulator will use the Twiss formula.

Twiss Formula

$$C_y = 2kt(Y + Y^H)$$

Where:

Cy = Noise Correlation Matrix

k = Boltzmann Constant

t = Temperature

Y = Y Parameter Matrix

H = Hermitian Matrix (complex conjugate transpose)

The **FBLOCK** file parameter is searched with the same methodology as searching library files, see "Search path priorities" on page 694. This means that if the FBLOCK file is not found in the current directory, the library where the corresponding FBLOCK instance was found is searched first if FBLOCK was actually read from a library. If not found, the directories are searched in the order specified by the option **SEARCH**.

Examples

```
.model dio D rs=4.68 bv=6.1 cjo=246p
.model Fblock macro lang=c
vin 1 0 dc 5 ac 1 pulse(0 5 1n 1n 1n 5n 10n)
rin 1 2 50

ytline Fblock param:
+ force_passivity=1
+ string: C:\s-parameterdata\lowpassfilter.s2p
+ pin: 2 0 3 0

dout 3 0 dio
.ac dec 10 1 10meg
.tran 10n 100n
.plot ac vdb(2) vdb(3)
```

```
.plot tran v(1) v(2) v(3)
.end
```

In the above example, we define a block ytline. By default, the CPF method runs. Delay extraction is allowed (if feasible), fit is set to individual, the model is assumed symmetric, passivity enforcement is set for pre-fit stage; refit, extrapolation to DC, and high precision flags are disabled, and the parameters **MAXCOL**, **MAXROW** are set to their default values, 1500 and 40,000 respectively. The file name is given in conventional form, by using the keyword "**string:**".

```
.subckt sparam_2p p1 p2 grnd
.model Fblock macro lang=c
y2port FBLOCK param:
+ idx_f=4
+ idx_m=1
+ no_delay=1
+ pin: p1 grnd p2 grnd
.ends sparam_2p
```

In this example, a block y2port refers to an S-parameter file *sb4.par* (since **idx_f**=4). Here, the S-block is described as a subcircuit. The DSP method is chosen and delay extraction is prevented.

Touchstone Data Format

The Touchstone® data format file is an ASCII text file in which data appears line by line: N lines for each data point of N ports. The data points are stored in increasing order of frequency.

The first of these N lines consist of a frequency value and N pairs of values for S, Y, Z, G, H, T or A parameters.

The (N-1) following lines contain N pairs of values.

Values are separated by one or more spaces or tabulations.

Tip: Touchstone data format files follow general syntax rules. The standard is available from the EDA Industry Working Groups website:
http://www.eda.org/pub/ibis/connector/touchstone_spec11.pdf

Example of S parameters for three ports:

```
F  SR11  SI11  SR12  SI12  SR13  SI13
   SR21  SI21  SR22  SI22  SR23  SI23
   SR31  SI31  SR32  SI32  SR33  SI33
```

Note

Two ports may also represented in single line format but will have a different parameter order (notice that **S12** and **S21** are swapped), see below.

Two ports on a single line:

```
Freq S11  S21  S12  S22
```

Two ports on dual lines:

```
Freq S11  S12
     S21  S22
```

Comment lines begin with an exclamation mark (!). The first un-commented line in the file must be a specification line. An optional specification line begins with the number symbol (#) followed by a space. Then, several optional parameters are specified in the following order:

- Frequency Unit (Hz, kHz, MHz, GHz). Default value is Hz.
- Parameter type (S, Y, Z, G, H, T, A). Default value is S.
- Data format (MA, DB, RI). Default value is RI. MA means Magnitude in Volts, and the phase in degrees. DB means Magnitude in dB, and phase in degrees. RI means Real and Imaginary parts.
- Reference impedance of each port (when all the ports have the same reference impedance, only one may be specified). Default value is all ports with the same 50 Ω reference impedance.

```
# [Hz|kHz|MHz|GHz] [S|Y|Z|G|H|T|A] [RI|MA|DB]
+ [R Val|R1 Val1 ... Rn Valn]
```

- The two-port noise parameters (NFMIN, GAMMA_OPT_MAG, PHI_OPT, RNEQ) can be used when you have specified a **.NOISE** command in the netlist and when the circuit to be analyzed is a two-port circuit. NFMIN is the minimal noise figure of the two-port. GAMA_OPT is the magnitude of the optimal reflection coefficient associated with the minimum noise figure. PHI_OPT is the angle of the optimal reflection coefficient associated with the minimum noise figure. RNEQ is the equivalent noise resistance.

Example

```
# khz s ri r 50
```

Frequency values are in kHz, the data are S-parameter data, they are stored in the format Real and Imaginary part and the reference impedance is 50Ω for each Port.

Mixed Mode S-Parameter Extraction

When extracting mixed mode S parameters the contents of the Touchstone output data file will change. For example the file header for a 2-port network may appear as follows:

```
! Data from foo
```

```
! S11 = SDD11
! S12 = SDS12
! S13 = SDC11
# HZ S RI  R1 1.000000E+01  R2 1.000000E+00
```

Chapter 14
Speed and Accuracy

Introduction

This chapter describes the algorithms in Eldo and their control options. Most of the information relates to the transient analysis, as it is the most time-consuming analysis, and also the most frequently used analysis.

The most relevant trade-off that users are interested in is the speed/accuracy trade-off. Circuit-level transient simulation is indeed the numerical resolution of an algebraic differential system of equations. As such, it is not an exact procedure (unlike the linear AC or NOISE analyses), and many 'switches' can be used to control the accuracy of the results. Usually, improved accuracy comes with an increased CPU time.

Eldo includes three different algorithms, namely NR (Newton-Raphson), OSR (One Step Relaxation) and IEM (Integral Equation Method). Each of these algorithms has its own set of properties.

- NR is the most general and robust algorithm, and it is always used by default. It is very accurate, and applicable to all kind of circuits, without restriction. NR is used by the vast majority of 'SPICE' commercial simulators, because of its generality.
- OSR is efficient for the analysis of large, loosely-coupled circuits, typically digital CMOS. It is reasonably accurate, and the CPU time grows almost linearly with the size of circuit, whereas the CPU time growth rate of NR is super-linear. However OSR works really well only if the loose-coupling assumption is verified, thus it is much less general than NR. For example OSR is not effective with bipolar circuits. Further details on OSR can be found in "OSR Algorithm" on page 1073.
- IEM is yet another completely different algorithm, unique to Eldo. It is useful when very high accuracy is desired and/or when NR shows stability issues. Some components cannot be formulated in a way that is compatible with IEM, thus it is also less general than NR. Further details on the use of IEM can be found in "IEM Algorithm" on page 1072 and "Integral Equation Method (IEM)" on page 1099.

Unless explicitly triggered, by the user with commands and/or switches in the netlist, neither OSR nor IEM are used by Eldo. By default, only NR is used, for the whole circuit.

Eldo is also able to partition a circuit into different parts that are simulated with different algorithms. For example, some partitions can be simulated with classical NR, whereas others are simulated with OSR or IEM. Each partition can use its own accuracy control parameters.

Speed and Accuracy in Eldo

Eldo has numerous control parameters to choose the most appropriate speed/accuracy compromise. This chapter attempts to present these parameters, covering the most important ones, and some of the less important ones.

The system of equations that represent the behavior of a circuit cannot be solved analytically, apart from trivial cases. Thus a simulator has to use numerical algorithms to find an approximate solution.

In the case of transient analysis, the problem to solve is to find the solution of a DAE (Differential Algebraic Equations) system. To simplify, and deliberately using loose notations, the 'solution' that the simulator tries to find is a set of N voltage waveforms $v_n(t)$, where n ranges from 1 to N, N being the number of nodes in the circuit, and t represents the time variable. These voltages are the solution of f(v(t), q'(v(t)), t)=0, where f() and q() are non-linear functions. Equation f()=0 is nothing but the expression of the Kirchoff law, i.e. the sum of all currents entering a node is null, at any time. Function q() models the dynamic part of the circuit, i.e. the generally bias-dependent charges in the circuit.

This system is differential and non-linear. To solve it numerically, time is discretized, and the equations are solved at discrete time points. The time-derivatives are approximated using a so-called integration scheme or method, using current and previous values of the solution $v(t_i)$, $v(t_{i-1})$,…, $v(t_{i-m})$. The number m of previous time points used to approximate the time-derivatives depends on the 'order' of the integration method. In all cases, an approximation error is introduced in this process.

Once the time-derivatives have been eliminated, the system to solve is 'simply' non-linear. Many numerical methods exist to solve this numerically. They are usually requiring a certain number of iterations, starting from an initial guess v_0, and each iteration providing, hopefully, a better estimate v_j of the solution v_{exact}. This iterative process will normally converge. Some criterion are needed to decide when to stop the iterative process and accept the last estimate as the 'solution' at the current time point. Again an approximation error is introduced here. One of the commonly used methods to solve non-linear systems of equations is the so-called Newton-Raphson method. It is commonly used mainly because of its generality and also for its relative robustness.

As a summary, mainly two types of errors are involved in the resolution of the system:

- Errors due to the numerical integration process
- Errors due to the numerical resolution of non-linear equations

Integration Methods

As explained before, the process of 'eliminating' the time-derivatives in the original circuit equations is a source of error. These time-derivatives are replaced with finite differences, which only approximate the true time-derivatives. One of the most basic approximation scheme is the Backward-Euler method, which approximates $v'(t_i)$ using the finite difference $(v(t_i)-v(t_{i-1})) / (t_i-t_{i-1})$. Intuitively, it is easy to understand that the smaller the time step $h=t_i-t_{i-1}$, the smaller the error. More sophisticated schemes exist, which provide less error with the same time step. For example the so-called trapezoidal and Gear methods are such methods.

The numerical resolution of differential equations is a vast subject, and there exists abundant literature about the subject. Dozens of methods have been proposed and studied in depth, some of them having very attractive properties, particularly allowing to use very large time steps without running into stability issues. However, in the context of circuit simulation, many of these methods are not practically applicable. The differential equations that model an electrical network (either IC or PCB) dynamics have unfortunately ‘bad’, undesirable characteristics. For example, they are implicit (in most if not all formulations used in commercial simulators) and very often ‘stiff’ (which means that the individual time constants involved in the system can routinely exhibit orders of magnitude differences). The cost of evaluation of the non-linear functions is also very high. This unfortunate situation leaves very few good practical candidates as integration methods.

Eldo implements three methods, namely the simple Backward-Euler (BE) method, the trapezoidal method (TRAP) and the Gear method (GEAR). The trapezoidal method is the default in Eldo. It provides a very good speed/accuracy compromise. TRAP and GEAR are both second-order method, whereas BE is a first-order method. The accuracy of BE is theoretically one order of magnitude worse than TRAP or GEAR, so it is usually reserved for cases where speed is the most important criterion, and accuracy can be somewhat sacrificed. In many cases, the speedup obtained with BE is not ‘spectacular’, although the loss in accuracy can be significant. This is because the GEAR and TRAP methods are still relatively simple methods (the derivative approximations are not too complicated). Thus the accuracy improvement with TRAP or GEAR over BE is generally worth the extra cost of CPU time.

Although TRAP and GEAR have similar theoretical accuracy, they still have their own characteristics. TRAP has the very undesirable tendency to generate numerical ‘ringing’, particularly on the current variables. Ringing shows up as obviously non-physical oscillations of small (or large!) amplitude riding on top of a correct and accurate average value. The oscillations have the rhythm of the time steps, they don’t belong to the circuit intrinsic time constant(s). They usually dampen over time, if the simulator properly controls the time step. This does not happen systematically with all test cases, but it is a well-known artifact of the TRAP method. All simulators, including Eldo, implement some code that tries to eliminate or reduce these oscillations, but sometimes, it may be very difficult to get rid of them entirely.

From the user point of view, thee are not many options. Clamping the maximum allowed time step is usually the most radical solution. However this may slowdown the whole simulation a lot. Increasing the accuracy may also help.

If these oscillations happen, and they are a problem (for example because the testbench attempts to measure currents with high accuracy), then the solution is to switch to the GEAR method. Much cleaner current waveforms will be produced by GEAR. However, everything has a price, and the price here is that GEAR has its own undesirable property to artificially damp natural oscillations of the circuit. Thus, for the analysis of pure oscillators, or when trying to identify local or global instability problems in a design, GEAR is not recommended, because it will tend to artificially ‘stabilize’ any circuit. BE is even worse with that respect.

If you want to get a feeling of how these methods behaves with respect to damping of natural oscillations, try the following: create a pure parallel LC network between a node you will initialize at 1Volt (using a .IC statement) and the ground (0). Pick L and C and the transient analysis duration T so that you can see about one hundred periods of the waveform (T=100.sqrt(L.C).2.pi). Try it with TRAP, then with GEAR, then with BE.

Time Step Control

As explained in the background section, the resolution of the system of equations uses discrete time steps, and errors are unavoidable in all realistic cases. Time step control designates the set of methods used by the simulator to select these time steps, so that the accuracy of the solution is maintained within predefined tolerances.

Time Step Control Algorithms Overview

To simplify, the time step can be controlled in three different ways. Eldo can use either:

- Local Truncation Error control (LTE)
- Rate-of-change control (DVDT)
- Iteration Count control (IC)

Some variants on these schemes are also available and detailed below, but these are the main three strategies.

By default, Eldo controls the local truncation error (LTE), and determines the time steps it can take based on estimations of this error. When a solution at time t has been accepted, to progress in time, Eldo will compute the value h of the largest time step it can take while still maintaining an acceptable LTE. Note that this is just a guess. The solution at time point t+h is predicted using the previous solutions at the previous time points. This serves as the initial guess. Then Eldo tries to achieve convergence at the new point t+h. If convergence cannot be achieved, the time step is reduced using a smaller time step h’ (h’<h), and a new resolution is attempted. If convergence is achieved, an estimate of the LTE is computed. If it is acceptable, the solution at time t+h is accepted, and a new cycle begins. If the error is found to be too large, then the time step is reduced in a way that should satisfy the truncation error constraint, and a new solution is computed. This process is reiterated until a time step and solution satisfying the truncation error criterion is found.

Thus when using truncation error control, there are two reasons why the time step may have to be reduced in the course of the transient simulation. Time step reduction can occur either because the Newton iterations do not converge, or because the estimated error on the time-derivative expression is too large. These two dimensions (Newton convergence and LTE) are usually highly intricate.

To monitor this, a very useful option of Eldo is the NEWACCT option. If .option NEWACCT is set, Eldo reports interesting (and readable) statistics about the iteration counts, average number of iterations per time step, number of time step rejections, and so on. This can be used to double-check whether the simulator runs ‘normally’ or not. These statistics are reported at the very end of the output .chi file. Eldo also reports some statistics in the original SPICE style, but the newacct reporting is much more readable and useful.

Convergence failure in the Newton iterations has several possible reasons. The initial guess may be simply too distant from the solution. This might happen if the chosen time step is ‘over-optimistic’ or if a sharp change in the circuit’s state occurs within the time step. Note that the simulator anticipates sharp changes in the stimuli, and all ‘break’ points such as the edges in PWL or PULSE signals force coinciding time points. Another reason for convergence failure can be discontinuities in the model equations, or simply strong non-linearities.

Controlling the local truncation error (i.e. the error made while approximating the time derivatives with finite differences) is the most conservative and rigorous way to control the time steps. It will usually provide more reliable and accurate results. This is the method selected by Eldo by default.

The rate-of-change control method uses the rate of change of the voltages to control the time steps. The idea behind this method is to control the time steps used so that the voltages do not change ‘too fast’. It is simpler than the LTE method, but also less accurate. There is no direct general relationship between the rate of change of the voltage and the actual truncation error, at least not under all conditions. The method can however provide accurate results if the rate of change is forced to remain small enough.

The iteration count method attempts to control the time step by monitoring only the rate of convergence of the Newton iterations. The idea being that if convergence is obtained rapidly, with just a few iterations, it probably means that the initial predicted guess was ‘good’, and conversely if many iterations are required, it probably means that the guess was incorrect. There is no attempt to estimate the truncation error. The control is entirely indirect, through the monitoring of the iteration count. This method is the least reliable of all and not necessarily any faster.

When selecting the time points, Eldo may also use internal heuristic rules, and, for example, adjust the time steps depending on the types of devices, the way they are connected, the scale of the simulation time, and so on, in order to obtain optimal results given the requested tolerances.

As a consequence there is no guarantee that from one release to another, the exact time point locations, or the time point density (local or global) will be the same. If the time points that Eldo selects naturally are not convenient for one reason or another, users must add explicit control options to alter this density. This can be through options such as: HMAX, INTERP, OUT_STEP, OUT_RESOL, and so on. See further details in this chapter, and also the command ".PLOT" on page 791.

Time Step Control—Algorithm Selection with the LVLTIM Option

To select one of the time step control methods which we described previously, the option **LVLTIM** is used. It can be set as an option in the netlist:

```
.option lvltim=0|1|2|3
```

> **Note**
> If you are not comfortable with these notions, you should simply leave everything by default.

- **LVLTIM**=0

  With **LVLTIM**=0, Eldo controls the time step based on the rate of convergence of the Newton iterations. This is called "Iteration Count" (IC) control. The time step control algorithm in this case is really simple and only depends on three parameters, namely **FT**, **ITL3** and **ITL4**. The default values of these control parameters can be altered with .option statements. The algorithm is as follows:

  If convergence is obtained in less than **ITL3** iterations, then the time point is accepted, and the next time step is increased by a factor 2 at most. If convergence is obtained in less than **ITL4** iterations then the time point is accepted, and the next time step is kept unchanged. If more than **ITL4** iterations would be needed, the time step is reduced (multiplied by **FT**), a new prediction is made, and then a new attempt to reach convergence begins.

  As apparent from the above, no estimation of the truncation error is calculated. This algorithm is usually faster, but it is also the less reliable.

  Default values are **FT**=0.125, **ITL3**=3, **ITL4**=13.

> **Note**
> If you decide to experiment with LVLTIM=0, it is highly recommended to start with the default values of **FT**, **ITL3** and **ITL4**, and to be careful when altering these parameters.

- **LVLTIM**=1 **DVDT**=0

  This triggers the rate-of-change time step control. The time step is adjusted depending on the parameters **ABSVAR** and **RELVAR** and not on the local truncation error estimation. The time step is controlled so that no voltage changes by more than **ABSVAR** (in absolute value) nor by more than **RELVAR** (in relative change). The default value of **ABSVAR** is 200mV. The default value of **RELVAR** is 0.15.

- **LVLTIM**=1 **DVDT**=–1

  This hybrid variant actually uses the same algorithm as in the case of the default **LVLTIM**=2, except that time steps are not rejected when truncation error is too large. In other words, truncation error estimation is used only to predict the time step that can be used, but if the iterations converge, the time step is accepted without LTE control. The simulation may be faster though less accurate.

- **LVLTIM**=2 (default)

  This is the default time step control algorithm. It is by far the most reliable algorithm, and yields the highest accuracy. In this mode, Eldo estimates the local truncation error (LTE) and adjusts the time step accordingly. The LTE estimate is used for both time step prediction and rejection control.

  A prediction of the largest time step is made, based on the LTE estimation of the previous time point. If convergence cannot be reached with this time step, the time step is reduced (divided by **FT**), a new prediction is made, and then a new attempt to reach convergence begins. If convergence is reached, the time point is not accepted right away. Instead, a new estimation of the LTE is calculated. If the LTE is within the tolerance, the time point is accepted (i.e. all state variables are updated, and Eldo proceeds with the next time point. If the estimated LTE is too large, the time step is rejected, and a smaller time step is retried.

  The truncation error can only be estimated, and the estimation algorithms are different depending on the integration algorithm (BE, TRAP or GEAR). It happens that many times, the truncation error estimate provided by the estimation algorithm/formula is significantly larger than the exact error (this has been experimented with simple circuits for which the differential equations can be solved analytically, and thus the exact error can be computed). Thus, in most circuit simulators – and Eldo makes no exception - a correction factor can be used to compensate this ‘over-estimation’ and not force unnecessary small time steps.

  The compensation of the truncation error is controlled with the **TRTOL** parameter. This parameter can range from 1 to 7 (the default value is 7). As **TRTOL** is made smaller, the estimated truncation error is made larger, and thus the time steps are globally reduced.

  Note than in 99% of cases there is absolutely no reason for you to play with **TRTOL**. The default value is correct for the vast majority of circuits and conditions.

  In general, the acceptable LTE threshold is controlled by the global EPS parameter. More on LTE control can be found in “Control of the Local Truncation Error (LTE)” on page 1068.

- **LVLTIM**=3

  With **LVLTIM** set to 3, the time step is controlled in the same way as for **LVLTIM**=2, i.e. LTE estimates are used to predict and control the time steps. However, additional time points are calculated based on the **TPRINT** value in the **.TRAN** statement. For example if using:

  ```
  .option lvltim=3
  .tran 1n 1u
  ```

  the time step control is based on the same LTE considerations as with **LVLTIM**=2, but additional time points are forced every 1ns, regardless of any LTE considerations.

  Obviously, if the TPRINT parameter of the **.TRAN** statement is very small, this will slow down the simulation significantly compared to **LVLTIM**=2, but will probably improve the accuracy (there are not many examples where computing extra time points can reduce accuracy). If on the other hand, the TPRINT is anyway larger on average than what is required from an LTE point of view, this will have little impact on speed, and may be useful for post-processing or FFT purposes.

Control of the Local Truncation Error (LTE)

When **LVLTIM** equals 2 or 3 (see previous section), LTE estimates are used to monitor the time step selection. Truncation error is the part of the solution error which originates from the approximation of the time-derivative terms (for example, in the case of a simple capacitor, the dv/dt term in the I=C.dv/dt equation) with finite-differences expressions. LTE can be estimated and monitored using the dv/dt terms or the v term itself.

When using LTE control to determine the acceptable time steps, Eldo may use the voltage quantities and/or the charge/flux quantities.

By default Eldo uses voltage quantities for LTE estimates. There are however situations where LTE on charge/flux is used:

- If **.OPTION QTRUNC** is specified, LTE will be computed on charges/flux, not on voltages. This corresponds to the way SPICE operates. This is sometimes more efficient and/or accurate for PCB applications, but often seems rather indirect to IC designers.
- If Eldo detects ‘PCB-like’ devices such as those shown in the list below, then LTE will be computed on charges/flux:
    - Current sources above 1A
    - Voltage sources above 500V
    - Inductors above 0.1H
    - Negative capacitors
    - Magnetic core devices

If option **QTRUNC** is set, LTE is computed using charges and flux quantities. In this case parameters (resp.) NGTOL, CHGTOL and FLUXTOL designate the absolute tolerances on (resp.) voltages, charges and fluxes, and RELTRUNC is the associated relative tolerance.

More about time step control

Eldo can also ‘clamp’ the time step to both a minimum value and a maximum value. To clamp the maximum time step you want to allow, use the HMAX option. Eldo will perform all its normal time step control as explained previously, but still not take any step larger than HMAX. Use this option carefully, as setting HMAX to a smaller value than required may force long simulation times. Unless very clear suspicion exists that the result is corrupted because the selected time steps are too large, this option should not be used.

To clamp the time step to a minimum value, use the HMIN option. This is a rather dangerous option, because it basically prevents Eldo to use the time step it should use to maintain the accuracy within the specified tolerances. Unexpected results or failures are possible if using an inappropriate HMIN value. The default value for HMIN is 1ps. Unless you are desperately looking for speed improvements, it is best not change HMIN.

To control the ‘acceleration’ of the time step when convergence is easily obtained, use the HACC option. The overall goal of Eldo is to take the largest time step possible, while still providing results that are within the requested tolerances (typically specified with EPS, RELTOL, etc.). Every time a time point has been computed and accepted, Eldo will perform some analysis and computations to determine what the next optimal time step should be for the next time point. If convergence is easily obtained (that is, with very few iterations) and if the LTE constraints are verified with a large margin, Eldo will attempt to take a larger time step for the next time point by multiplying the current time step by a certain acceleration factor, as specified by HACC. The default value for HACC is 2, which means that Eldo will attempt to multiply the current time step by 2 at most.

Specifying a value with option HACC overrides the default value of 2. This time step acceleration strategy is a compromise. Choosing a conservative value (1.1x for example) would provide very little speedup, but the next time point would most likely easily converge. Alternatively, choosing a large factor (10x for example) when convergence is easily obtained is tempting, and would in theory provide a significant speedup. However, chances are high that the time point computed with the new (too) large time step will either not converge because of non-linearity, or will have to be rejected because the LTE constraints will be violated. In these cases, Eldo would then step back and recompute a new time point with a smaller time step (see option FT). Therefore, an aggressive value might be a worse choice than a conservative one. The default value, 2, reflects this compromise and is a robust choice providing the best results on average. Unless you have very good reasons to do so, you are discouraged from changing the value of HACC.

Using a fixed time step

A fixed internal time step can be specified by the **`STEP`** parameter; this may be useful when simulating certain classes of circuits (for example switched-capacitor circuits), however, care must be taken. Overall, it is not safe, and may cause excessively long simulation times.

If the requested **`STEP`** value is unreasonably small, the simulator may be fooled into believing that some nodes are not changing and will set them into “latency.” A badly chosen fixed time step can thus sometimes have unwelcome results.

A **`STEP`** value that is too large may similarly cause problems. Consider a bipolar circuit when a large transition of an input signal occurs between two steps. This will nearly impossible to handle for the simulator.

Overall, using this option in the case of regular IC circuit analysis is highly discouraged.

Newton Iterations Accuracy Control

The main parameters controlling the accuracy of the Newton iterations are **RELTOL**, **VNTOL, ABSTOL** and **CHGTOL**

Whenever Newton iterations are used to solve the non-linear system of equations, the following convergence criterion for voltages, currents and charges are used:

- For node voltages:

  |V(i) - V(i-1)| < **RELTOL***|max(|V(i)|, |V(i-1)| + **VNTOL**

  where V(i) is the value at current iteration i and V(i-1) the value of the previous iteration.

- For branch currents:

  |I(i) - I(i-1)| < **RELTOL***|max(|I(i)|, |I(i-1)| + **ABSTOL**

  where I(i) is the value at current iteration i and I(i-1) the value of the previous iteration.

- For charges:

  |Q(i) - Q(i-1)| < **RELTOL***|max(|Q(i)|, |Q(i-1)| + **CHGTOL**

  where Q(i) is the value at current iteration i and Q(i-1) the value of the previous iteration.

The same RELTOL parameter controls the relative tolerance for voltages, currents and charges. However, when voltages, currents or charges become 'small' in absolute value, a relative tolerance becomes useless, thus absolute tolerances are required. Of course the typical orders of magnitude of voltages, currents and charges are quite different, thus the necessity to have specific absolute tolerances (VNTOL, ABSTOL, CHGTOL).

As a rule of thumb, if using the default settings, a simulation that goes well should use at most three or four Newton iterations per time step. This can be less if the circuit is almost linear, or if the settings such as the RELTOL value are relaxed. If many more iterations are needed, it means that either the time step control options have been relaxed a lot (and potentially the truncation error is large), or there are strongly non-linear characteristics in the circuit, which are difficult to solve. The number of Newton iteration will also increase if the RELTOL value is reduced to obtain more accuracy.

Global Tuning of the Accuracy—EPS

A global parameter, **EPS**, is used as a general controller to set the required accuracy of a simulation. When this parameter is changed, a collection of lower-level parameters are adjusted accordingly, affecting both the Newton convergence tolerances and the time step control tolerances.

Is is always possible to set these low-level parameters individually. However using EPS guarantees that the adjustments to the low-level parameters are consistent, which is not always easy to achieve for beginners or users not too familiar with circuit simulation tricks…

The parameter adjustments induced by EPS are shown in the table below (the global TUNING parameter which appears in the first line of the table is explained in the next section "Global Tuning of the Accuracy—TUNING" on page 1071).

Table 14-1. Global Tuning of Accuracy

TUNING			STANDARD			ACCURATE		VHIGH	
EPS[1]	1e-1	1e-2	**1e-3**	1e-4	1e-5	**1e-6**	1e-7	**1e-8**	1e-9
RELTOL	5e-2	5e-3	**7.5e-4**	5e-4	2.5e-4	**1e-4**	5e-5	**1e-5**	1e-6
VNTOL	1e-6	1e-6	**1e-6**	1e-6	1e-6	**1e-6**	1e-7	**1e-8**	1e-9
CHGTOL	1e-9	1e-9	**1e-14**	1e-14	1e-14	**1e-15**	1e-16	**1e-18**	1e-18
FLUXTOL	1e-11	1e-11	**1e-11**	1e-11	1e-11	**1e-11**	1e-11	**1e-11**	1e-11
ITOL	1e-6	1e-6	**1e-6**	1e-6	1e-6	**1e-6**	1e-6	**1e-6**	1e-6
ABSTOL	1e-12	1e-12	**1e-12**	1e-12	1e-12	**1e-12**	1e-13	**1e-14**	1e-15
RELTRUNC[2]	5e-2	5e-3	**7.5e-4**	5e-4	2.5e-4	**1e-4**	5e-5	**1e-5**	1e-6
CHGTRUNC[3]	1e-9	1e-9	**1e-14**	1e-14	1e-14	**1e-15**	1e-16	**1e-18**	1e-18

1. See .OPTION EPS for further information.
2. When set indirectly though EPS, RELTRUNC always gets the same value as RELTOL. RELTRUNC is only used if LTE is controlled on charges/flux, i.e. when .option QTRUNC is set.
3. When set indirectly though EPS, CHGTRUNC always gets the same value as CHGTOL. CHGTRUNC is only used if LTE is controlled on charges/flux, i.e. when .option QTRUNC is set.

Setting EPS automatically changes the low level parameters shown in the table above. However it is still possible to override some of them. For example, if using `.option eps=1e-3 reltol=1e-8`, RELTOL will be set to 1e-8, instead of 1e-3. VNTOL will be set to 1e-6, indirectly though EPS. This would not be too consistent probably, but it is still allowed.

It is important to notice that RELTOL, which is one of the main parameters defining the global accuracy, does NOT scale linearly with EPS.

Global Tuning of the Accuracy—TUNING

Another global parameter, **`TUNING`**, may be used as a general controller to define the accuracy of a given simulation. With TUNING, the user does not provide numerical values for the accuracy control switches. Instead, a qualitative flag is specified for the desired accuracy, and similarly to the effect of EPS, a number of adjustments are activated. Actually, 'accuracy' in this context should be understood as 'speed/accuracy compromise'.

The TUNING parameter can take four values, namely FAST, STANDARD, ACCURATE and VHIGH.

The names are self-explanatory. FAST can be used when the number one concern is to accelerate a simulation, and you are ready to sacrifice some accuracy to achieve this.

The FAST setting is equivalent to EPS=1e-3, overridden with VNTOL=10e-6, ABSTOL=100e-12, RELTOL=1.25e-3 and CHGTOL=1e-12.

Thus FAST is not entirely equivalent to any given value of EPS. It is an adequate setting if you are not too worried about picoAmps and nanoVolts accuracy, and you want mostly a short runtime.

STANDARD, ACCURATE and VHIGH are completely equivalent to specific values of EPS (see Table 14-1 on page 1071 in the previous section).

STANDARD corresponds to the Eldo default settings, i.e. is equivalent to setting EPS=1e-3. If you do not set any option in the netlist file, this is what Eldo will use. Thousands of test cases covering all possible IC technologies have shown that it was the best compromise for what Eldo tries to achieve by default, i.e. reliable and accurate results in the shortest CPU time.

Finally ACCURATE and VHIGH (which stands for Very HIGH accuracy) will alter the tolerance switches to achieve higher degrees of accuracy, usually to the expense of longer simulation times of course. ACCURATE is equivalent to setting EPS=1e-6, and VHIGH is equivalent to EPS=1e-8.

The VHIGH setting must be used carefully, as it can lead to excessively long simulation times. It is however sometimes necessary, particularly for the cases of the startup phase of sensitive oscillators.

Using EPS settings smaller than 1e-9/1e-10 is usually un-reasonable and not recommended.

Setting the TUNING flag simply uses the .OPTION mechanism. For example:

```
.option TUNING=ACCURATE
```

IEM Algorithm

With IEM, the differential system is first transformed into a system of integral equations, which is then transformed into an algebraic system by series expansion. The truncation error results from the finite number of terms in the series. In all cases, truncation error is also a function of the time step size. The IEM method is described in more detail in a dedicated chapter Integral Equation Method (IEM).

Once the non-linear algebraic system is obtained, it is solved by an iteration loop. IEM targets improvements of the accuracy of the solution (compared to Newton). It does not specifically target large circuits, so it is mostly used for cell of macro-block simulations where accuracy is critical.

IEM is interesting for cases where high accuracy is required and/or when the default Newton method runs into numerical stability issues due to the integration methods (trapezoidal ringing for example). IEM however does not support all devices, macro-models, and so on, that Newton supports. Some devices cannot be efficiently formulated in a way that is compatible with the IEM algorithm. Thus the applicability of IEM is somewhat limited. It is mostly used for cell characterization applications. The Integral Equation Method (IEM) chapter describes the limitations of the IEM method.

Integration method

When using IEM, the notion of integration method like BE, TRAP or GEAR, is irrelevant. The equations are cast to an integral form prior to the resolution. Thus there are no choices about a numerical integration method to use.

Time step control

When using IEM, the time step control algorithm uses local truncation error (LTE) control. Only the LVLTIM=2 and LVLTIM=3 methods are selectable together with IEM. The other methods are not reliable enough when used together with IEM, and they will be refused by Eldo.

Accuracy control

When using IEM, the same accuracy control parameters as those used for the default Newton method are available. Thus mainly the RELTOL and ABSTOL parameters will control the accuracy of the resolution

OSR Algorithm

The One Step Relaxation (OSR) algorithm of Eldo is dedicated to the simulation of large MOS circuits showing weak local couplings (large-scale feedback loops are not a problem). It is a unique algorithm to Eldo. Its primary usage is the fast transistor-level simulation of large digital CMOS circuits, for which the weak local coupling assumption is generally valid. When the conditions for its applicability are met, its speed and accuracy are excellent. In many cases, it is just as accurate as the Newton method, but much faster, and also the growth of the CPU time with the circuit size is almost linear, which allows simulating larger networks. Although its capacity is still less than that of fast-SPICE timing simulators such as ADiT, it provides a quite interesting point in the speed/accuracy plane.

OSR is not particularly efficient with tightly coupled analog circuits. Better results will be obtained using either the default Newton-Raphson or IEM.

OSR can be activated explicitly or indirectly. The most straightforward way to activate OSR is to use ".OPTION OSR". If this option is set, Eldo will attempt to simulate the whole circuit with OSR. However, if the circuit contains elements that prevent OSR from being effective, or if its connectivity is such that OSR will fail, Eldo will revert back to the default Newton algorithm, for part or all of the circuit. A warning message is then displayed.

Actually, Eldo is able to simulate a circuit with certain parts handled by ORS, and other parts handled by Newton. More details on this possibility are given in the section 'Combined OSR/Newton simulation'

Additional information related to each of the cases above can be found in the appropriate sections of "Simulator and Control Options" on page 939.

OSR and the notion of latency

When OSR is used, Eldo places the nodes for which the surrounding nodes do not show any voltage variation greater than **`EPS`** volts into 'latency'. Thereafter, Eldo effectively bypasses the calculation of such nodes. This allows significant CPU time savings, especially for large digital circuits where quite often only a fraction of the nodes are changing over one time step.

Due to the formulation of OSR, latency exploitation is very natural and effective. Latency is much more complicated to control in a reliable way in the context of the regular Newton method.

Even with OSR, latency and above all 'wake-up from latency' is however always tricky. When slowly varying signals are applied to a circuit with high capacitances, Eldo may not detect any voltage variation within one time step, resulting in nodes being placed in latency. Depending on the tolerances (EPS, this may cause incorrect simulation results. In case latency is potentially the source of problems, the **`.OPTION NOLAT`** command may be used to suppress the use of latency. If this option is set, Eldo will not use attempt to use latency, and will solve all nodes, whatever their activity.

Integration method

When using OSR, the same integration methods as for the case of Newton (namely BE, TRAP and GEAR) can be used (using **`.OPTION METHOD=`**).

Time step control

When using OSR, the default time step control algorithm is the same as in the case of Newton (LVLTIM=2), and thus it uses local truncation error (LTE) control. Only the LVLTIM=1, LVLTIM=2 and LVLTIM=3 methods are selectable together with OSR. The other method (LVLTIM=0 is meaningless in the context of OSR (there no Newton iterations when using OSR…).

Accuracy control

The accuracy control when using OSR is much simpler than with Newton. The global parameter **EPS** is used to control everything. Furthermore, the actual value of **EPS** (in Volts) is directly used for the convergence control.

Accuracy of the relaxation loop and the inner loop is directly controlled by **EPS**.

- Relaxation loop is stopped when:

  |V(irelax) - V(irelax-1)| < **EPS** for all nodes

- Inner loop

  |V(iter) - V(iter-1)| < **EPS**/50 on the current node

Combined OSR/Newton simulation

Eldo can ‘partition’ a circuit so that some blocks are simulated OSR and other blocks are simulated using the standard Newton algorithm. Typical candidate examples would include circuits with large digital CMOS blocks driving and/or driven by analog blocks such as voltage regulators, bandgap references, high-gain operational amplifiers, and so on.

In these cases, the speed and capacity of OSR is used to handle the large weakly-coupled sections of the circuits, and the accuracy and ability to handle tightly coupled devices of Newton is used for the analog blocks.

The partitioning can be left entirely to the discretion of Eldo, or the user can try to help and indicate which blocks (subcircuits) must be simulated with OSR or Newton. In this latter case, the partitioning is always ‘indicative’ only, and Eldo may choose to alter the boundaries of the partitions to accommodate the requirements of the OSR algorithm.

There are several ways to trigger these mixed OSR/Newton simulations:

- **.OPTION BLOCKS**=**NEWTON**

  This first partitioning method is automatic, based on the degree of coupling between devices. A global option is set (**.OPTION BLOCKS=NEWTON**). Eldo will identify the blocks consisting of tightly coupled devices, and place them in partitions that will be solved using the Newton method. The loosely coupled nodes are placed in the OSR partition. The global circuit is solved using a relaxation loop. ‘Simply’, the relaxation loop operates with individual nodes and also group of tightly coupled nodes (the Newton partitions). Note that the identification of tightly coupled devices is not based on the netlist hierarchy (**.SUBCKT**, and so on).

- Flag **(ANALOG)** given on **SUBCKT** definition

  This partitioning methods ‘tags’ the subcircuit definitions. The subcircuit definitions tagged with (ANALOG) will be solved using the Newton method. This is useful when the netlist hierarchy does reflect the nature and coupling level of the blocks. All nodes which are not inside user-defined **(ANALOG) SUBCKT**, and which are connected only to MOSFET, grounded capacitances, or low-value floating capacitances (the threshold value for floating capacitance is set with **.OPTION CAPANW=**), will be solved by OSR, all other nodes being solved by Newton. See the subcircuit definition syntax ".SUBCKT" on page 898.

- **.NWBLOCK** [**RELTOL**=val] [**VNTOL**=val] <list_of_nodes>

  This method is the most 'manual' one. Each Newton block is defined explicitly, with a list of nodes. Hierarchical names and wildcards in such names are allowed. Optionally the required Newton accuracy can be redefined for each block, using the **RELTOL** and **VNTOL** parameters. See the description of the **.NWBLOCK** option in the Simulator and Control Options for additional details.

With any of the partitioning methods described above, OSR is 'implicitly activated'. OSR becomes the default method, and Eldo (or the user) defines what still has to be simulated with Newton.

Whenever OSR and Newton are activated simultaneously, the OSR and Newton rules for accuracy control apply to the OSR and Newton partitions respectively. For example the RELTOL and VNTOL parameters still define the accuracy of the nodes inside the Newton partitions, whereas EPS defines the accuracy of the OSR nodes.

Simulation of Large Circuits

The Eldo circuit simulator is capable of handling very large circuits. To handle such circuits, containing several hundreds of thousands of components, memory usage becomes an important consideration.

For more information on memory requirements and dimensioning, please refer to the *AMS Installation Guide*.

For the simulation of large circuits, the following suggestions may help.

- Explore the manual...

  There are indeed several dozens options and features which can significantly impact speed, and help with the simulation of large circuits. Some of them are extremely specific and relevant only in the presence of such or such element. A good place to start is the Simulator and Control Options chapter, which lists all available .OPTION, grouped by categories. Particularly the Simulation Speed, Accuracy and Efficiency Options section complements the present chapter with reference data.

- Use workstation with sufficient RAM and SWAP.

  As with any software, an attempt to run a simulation with real memory far lower than that required to contain the circuit can result in a phenomena known as "thrashing."

When this occurs, CPU time is almost totally taken up in swapping data in and out of main memory, with little or no useful computation being made. Any tool that monitors process activity will report CPU usage, real memory usage and swap usage. When running large simulations, it is highly recommended to use of these tools to monitor the simulation processes.

- Use the 64 bit versions of Eldo.

  If more than 2 GB of real memory is needed, consider using the 64 bit versions of Eldo. This is activated by using “eldo -64b -i circuit.cir” on the command line.

- Specify node number via **`MAXNODES`** and **`MAXTRAN`** options.

  If you have an idea of the number of nodes and components (grounded capacitors not included) your circuit has, it is wise to specify these options:

  ```
  .OPTION MAXNODES=VALUE MAXTRAN=VALUE
  ```

  In this case, Eldo directly allocates the correct amount of memory, reducing the requirement of using the expensive C function `realloc()`.

- Collapse the intrinsic MOS transistor nodes

  When simulating large circuits containing mostly MOS transistors, the number of nodes grows very quickly with the number of MOS itself. By default, Eldo creates explicit internal nodes for the drain and source parasitic access resistors. Thus each MOS transistor with non-zero access resistors adds two internal nodes to the system. Although these nodes do not connect to anything else, and the sparse techniques used to solve the matrices greatly reduce the impact of these additional anodes upon the CPU time, they still contribute to create a much larger system. For example if 50,000 MOS are instantiated, the typical size of the circuit could be 30,000 or 50,000 ‘true’ nodes, but grows to 130,000 or 150,000 with two internal nodes per MOS transistor. Even for Eldo, this is quite a change in problem size.

  An option exists to ‘collapse’ these resistors into the drain current calculation, thus avoiding explicit creation of these intrinsic nodes. This option is **`NONWRMOS`**. With this option, the effect of the parasitic resistors upon the drain current is still taken into account, although not as accurately as when the nodes are explicitly created. Particularly, some of the overlap capacitances in the MOS model become connected to the external drain/source, whereas they really are connected to the internal nodes. This may change results slightly.

  However, experience has shown that the accuracy degradation is barely measurable, particularly when these resistors remain ‘small’. So, when having to simulate huge circuits, this is definitely something to try. It is however recommended to ‘calibrate’ the accuracy degradation caused by the option with a reasonably small circuit first. And then to decide whether this degradation is acceptable or not for the large circuit.

This option, as its name indicates, only affects MOS transistors. It has no effect upon the handling of the internal base, collector and emitter nodes of bipolar transistors, neither upon diodes.

When this option is active, there might be situations where DC cannot be found. If this happens, remove the option. Even when DC is found, if the netlist contains some MOS devices with forward biased bulk-source and bulk-drain junctions, Eldo will detect such situations during DC convergence, and will resimulate with the option disabled.

- Optimize the density of time points.

  Is not always possible to directly alter the density of time points that the simulator must compute to maintain a given accuracy. However it is highly recommended to get familiar with some of the options which affect the number of time points. Particularly, the OUT_STEP, INTERP, FREQSMP and OUT_RESOL might have very significant impacts. Detailed discussion of these options can be found in "Limiting the Size of Transient Output Files" on page 793.

- Use the Periodic Circuit Speedup (**PCS**) option.

  Used to increase the simulation speed for circuits with periodic or nearly periodic nature. PLLs in near-lock state belong to this category. The amount of speedup depends on the design nature. The speedup will be more significant with relatively large circuits. With circuits showing no periodicity at all, the option will not usually provide any speedup. Periodic Circuit Speedup is invoked by setting **PCS**=1|2 for BSIM3v3 models only (**PCS**=1|2 represent two possible speed optimization methods, **PCS**=1 is more recommended) or **PCS**=3 for BSIM4 and BSIM3v3 models (with same speed optimization as **PCS**=1). Default is 0. This option only supports the BSIM3v3 and BSIM4 models.

  BSIM4, unlike BSIM3, has many parasitic configurations: gate resistors, body resistors network, bias dependent access resistors, and so on. These parasitic configurations are controlled by a set of instance and model parameters (Rdsmod, Rbodymod and Rgatemod). As the complexity of the parasitic configuration around the core model increases, the gain expected by the PCS decreases.

  This option should not affect the accuracy of the results.

- Try suppressing DC analysis for large digital circuits.

  This is achieved by setting the **UIC** option of the **.TRAN** command. Very large circuits are often digital circuits for which an accurate and time consuming DC analysis may not be necessary. The logical initialization performed by Eldo prior to any analysis should ensure that the transient analysis that follows will start with nodes correctly preset. Initial voltage conditions are specified using the **.IC** command.

- Decrease the accuracy for MOS digital circuits.

  For MOS digital circuits, accuracy can often be decreased to 10mV instead of the default value of 5mV. This can save a large amount of CPU time.

- Employ the **.USE** command.

  It may be useful to split the circuit into a number of blocks, find DC solutions for each block and then find the DC solution of the circuit as a whole using the separate solutions as a starting point.

Tips for Troubleshooting and/or Improving Convergence and Performance

Operating Point Calculation

Before any kind of analysis is carried out by Eldo, a DC or operating point for the circuit in question is usually carried out, unless the UIC parameter is present in the .TRAN command.

DC and operating point convergence times can vary greatly, depending on the type of circuit simulated. Eldo provides a number of commands which may be used to speed up the convergence process. Each command is briefly described in the following subsections.

.IC V<NN>=VALUE

When used, Eldo fixes the specified node voltages for the duration of the DC analysis. If the **UIC** parameter is also present (in the **.TRAN** command), no DC analysis is performed and the voltages are initialized as defined in the **.IC** command. All other voltages on nodes not initialized in the **.IC** command are determined by Eldo itself. During subsequent analysis (transient), the node voltages are freed of their initial values, and may therefore assume different values.

See ".IC" on page 682 for further details.

.NODESET V<NN>=VALUE

This command is used to help calculate the DC operating point by initializing selected nodes during the first DC operating point calculation. After the first calculation has been completed the node values are "released" and a second DC operating point calculation is started. This command is useful when the whereabouts of the DC operating point is known, enabling the simulator to converge directly to it and also for bistable circuits or circuits with more than one operating point.

See ".NODESET" on page 743 for further details.

.GUESS V<NN>=VALUE

Enables the user to set the initial voltages to specific nodes for the first iteration of a DC operating point run.

See ".GUESS" on page 677 for further details.

.RAMP

This command can be used when no DC operating point has been found using the above methods. Two ramping facilities are provided.

DC convergence algorithms are automatically used by Eldo when the standard DC algorithm fails to converge. The sequence of algorithms used is as follows: Newton, Gmin ramping, DC ramping, Transient ramping, PSTRAN, T° ramping, and DPTRAN. The DPTRAN algorithm is used as a last attempt at convergence.

Note

See ".RAMP" on page 852, "`GRAMP=VAL`" on page 985, "`PSTRAN`" on page 1017, and "`DPTRAN`" on page 1017 for further details.

.OPTION VMIN, VMAX

By default, Eldo looks for the DC operating point within the limits of the circuit power supply. In the case of a circuit which contains elements that create energy, such as voltage or current controlled sources, it is possible that Eldo can look for solutions outside the expected bounds. If so, Eldo displays the message:

```
Searching operating point between [x1,x2]
```

The `x1` and `x2` limits can be controlled by the user in order to speed up the DC convergence process using the **`VMIN`** and **`VMAX`** parameters. It must be stressed, however, that unrealistic limits can also cause convergence problems.

.SAVE, .USE & .RESTART

A circuit may need to be simulated several times. In each case, the operating point must be determined although it may be the same for each run or may be only slightly different; in any case a large change of operating point is unlikely. To avoid the wasteful re-calculation of the operating point, a system is provided to store and reload the result of the DC analysis from the simulator; this significantly increases performance since operating point calculation time is thereby reduced.

The save, use and restart system is based on the commands:

```
.SAVE [FNAME] DC|END|TIME=VAL [REPEAT [ALT|SEQ]] [TEMP=VAL] [STEP=VAL]
+ [TYPE=NODESET|IC] [LEVEL=ALL|TOP] [CARLO=index]
.USE FNAME [NODESET|IC|GUESS|OVERWRITE_INPUT]
.RESTART FNAME [FILE=TMPFILE]
```

All commands listed above must be placed in the circuit description file.

The **.SAVE** command, as its name implies, is used to save analysis data to a file.

The **.USE** command is intended to be used to load DC or operating point information for one or more parts of the circuit to be simulated; its application is thus in cases where a large or complex circuit is to be simulated and the DC or operating point data for parts of the circuit is already known. Multiple **.USE** commands may be used in a simulation.

The **.RESTART** command is different from the **.USE** command in several ways. Firstly, whereas the **.USE** information may apply only to part of the circuit, **.RESTART** data always applies to the whole circuit. In fact, the **.RESTART** command first checks that the node names specified in the file used to "restart" the circuit, exactly correspond to those specified in the circuit description file where the **.RESTART** command is placed. The **.RESTART** command may be used to either load the operating (DC) point data and thereafter perform a transient simulation, or it may be used to restart a transient simulation from a specific point where the simulation had been interrupted in an earlier run. Obviously, to carry out a transient analysis from a specific point, the **.TRAN** command parameters need to be changed to specify the new simulation times.

Note

You only need to change the **.TRAN** command if you want to change its STOP time. You don't need to change its START time.

The following sections describe the use of the above. First, here is a short synopsis.

```
.SAVE [FNAME] DC|END|TIME=VALUE [REPEAT] [TEMP=VALUE] [STEP=VALUE]
.USE [FNAME]
.RESTART [FNAME] [FILE=TMPFILE]
```

- FNAME

  Filename, including extension, where the data is to be stored. If not specified, Eldo uses *<cirname>.iic* where *<cirname>* is the circuit filename.

- **DC**

  Save the DC data.

- **END**

  Save the complete simulation environment at the end of a transient analysis run, to allow a restart of the simulation from that point.

- **TIME**=VALUE

  Save the simulation status after the specified simulation time has elapsed.

- **REPEAT**

  Save the simulation status after each interval of simulation time has elapsed, as specified in **TIME**=VALUE. The save file is overwritten each time a run is saved. This is similar to the **.OPTION SAVETIME**=VALUE command, except that the latter command is in CPU time as opposed to simulation time.

- **TEMP**=VALUE

  Save the simulation status only when simulation temperature is equal to the value provided as parameter to **TEMP**. This is useful when sweeping through several temperatures.

- **STEP**=VALUE

  Save the simulation status when the **STEP** parameter value, which is being swept as the result of a **.STEP** command, equals the value specified in the **.SAVE** command.

- **FILE**=TMPFILE

  Binary output file. If the restarted simulation is to continue writing binary output data to a *.wdb* file, appending the new simulation data to the end of the previous *.wdb* file, then the previous file must be renamed or else it will be lost. The **FILE** parameter of the **.RESTART** command provides the new filename, TMPFILE, for the old *.wdb* file. The system then concatenates the old *.wdb* file data with the new *.wdb* data in the original *.wdb* file.

.SAVE DC in Combination with RESTART

The DC or operating point values for a circuit can be saved to a file and used later to greatly shorten this part of the analysis, when the circuit is re-simulated. This is achieved by employing the **.SAVE DC** and **.USE NODESET** commands, provided that circuit node names remain the same. The following steps are used, assuming that the circuit is stored in the circuit description file *<cname>.cir*.

1. Add the following line to the circuit description file:

   ```
   .SAVE <FNAME> DC
   ```

   If no filename <fname> is given, the default used is *<cname>.iic*.

2. Find the operating point of the circuit. This is stored in the file as specified in the **.SAVE** command.

3. Now change the circuit description file to include any other analysis required. Do not specify **dc**, **uic** or operating point calculation. Add to the circuit description file one of the lines:

   ```
   .RESTART  <FNAME>  NODESET       ! to use .NODESET or alternatively
   .RESTART  <FNAME>  IC            ! to use the .IC system
   ```

   The simulator will attempt to use the operating point information stored in the <FNAME> file.

To find out the differences between **.NODESET** and **.IC** strategies, see the "Operating Point Calculation" on page 1079.

.SAVE DC in Combination with USE

Eldo provides a system of combining the DC values of parts of a circuit to speed up the operating point calculation of the complete circuit. The DC state of the circuit sections which are available are stored in separate files. Such "circuit parts" may of course be whole subcircuits. In the general case, the DC data will be available on several files, called here `<OPDATA_SS>`. To "use" such for the DC calculation of a complete circuit, add the following lines to the circuit description file, and for each `<OPDATA_SS>` file:

```
.USE <OPDATA_SS> NODESET      ! to use .NODESET or
.USE <OPDATA_SS> IC           ! to use the .IC system
```

The simulator will attempt to use the operating point information stored in all the `<OPDATA>` files.

To find out the differences between **.NODESET** and **.IC** strategies, refer to ".SAVE, .USE & .RESTART" on page 1080.

.SAVE END in Combination with RESTART

It is possible to run a transient simulation and save the final state of the run. Thereafter, it is possible to restart the transient simulation from the time the last simulation was ended. To achieve this, the **.SAVE** command must be used in combination with the **END** parameter.

To restart the simulation, the circuit description file must be modified. The original **.SAVE** command must be removed and the **.TRAN** command must be modified to reflect the new end time. Furthermore, a **.RESTART** command must be included in the circuit description file.

```
.RESTART <FNAME>
```

Using the above procedure, the simulation will produce a new binary output file (*.wdb* file) which contains data starting from the end of the last simulation. If it is desired to concatenate old and new *.wdb* files, then the old *.wdb* file should be renamed to say `<TMPFILE>` and the above **.RESTART** command modified as follows:

```
.RESTART <FNAME> FILE=<TMPFILE>
```

The system then concatenates the old *.wdb* file data with the new *.wdb* data.

More Sophisticated SAVE & RESTART Procedures

The complete **.SAVE** syntax shown in the line below, includes provision to save the simulation state either periodically or at predefined times and/or temperature or sweep (**STEP**) parameter conditions:

```
.SAVE [FNAME] {DC|END|TIME=VALUE} [REPEAT] [TEMP=VALUE] [STEP=VALUE]
```

It is possible to save the result of the simulation periodically (with **REPEAT**), to save the run for a particular temperature (with **TEMP**) or for a particular swept parameter (with **STEP**).

The previous section contains a description of each of the parameters of the **.SAVE** command.

In each case, the rule to restart the simulation from a predefined state is the same. In each case, to restart the simulation, the restart command needs to be added to the circuit description file and, of course, the save commands contained therein should be removed.

The transient simulation parameters in the **.TRAN** command may also need to be modified to reflect the new desired simulation time stop.

Aborting simulation and saving current state

If a transient simulation run is interrupted by the user with a Control-C, Eldo prompts the user, inquiring if the simulation status should be saved. If the user answers in the affirmative, the simulation data is saved in a file called *<cirname>.sav* where `<cirname>` is the name of the circuit description file with no extension.

If the user wishes to restart the software from this interrupted point, then the circuit description file should be modified to include the command:

```
.RESTART
```

The simulation will then be restarted automatically from the time it was interrupted. The .TRAN command needs to be updated. Options and/or stimuli might be changed as well.

Chapter 15
Eldo Reduction

Introduction

Eldo allows you to verify design functionality and timing including the effects of physical layout. In order to include these effects of layout, you must generate a netlist that includes parasitics extracted from layout using an extraction tool such as xCalibre™ from Mentor Graphics. The extracted parasitic information is included in the netlist in the form of large networks of passive resistors, capacitors and inductors connected to the transistors or other active objects of the circuit. Due to the large number of elements that they contain, such parasitic networks strongly constrain post-layout simulation both in terms of memory and CPU time. The Eldo Reduction capabilities allow to cope with this constraint by susbtituting such networks on the fly by approximately equivalent but smaller sized networks computed by the TICER reduction engine[1] [2] [3]. The quality of the approximation can be controlled by setting a maximum delay error, a maximum noise error and/or a cut-off frequency as explained in this chapter.

Not only back-annotated circuits may benefit from the savings in computer resources allowed by the Eldo Reduction but also some specific circuits. For instance, this is the case for power distribution networks having large portions of purely resistive networks.

The reduction methods available in Eldo are also, and primarily, available in the xCalibre extraction tool, and it is strongly recommended to activate the reduction of parasitics while performing extraction with Calibre® xRC™/xL.

Nevertheless, even when the reduction of parasitic networks was performed in the extraction tool there may be some benefit in performing reduction in Eldo. This is clearly the case when the parasitics were generated to be valid over a frequency range that far exceeds the signal frequencies involved in the simulations to be performed. Using its reduction capabilities allows Eldo to take full benefit of reduction for each simulation individually while using parasitic networks valid over a possibly larger range of frequencies. Although this provides a convenient capability, it remains preferable and it is recommended to use different sets of extracted parasitics corresponding to the various frequency ranges of the simulations to be performed.

---

1. B. Sheehan, *Branch Merge Reduction of RLCM networks*, proceedings of the IEEE International Conference on Computer-Aided Design, 9-13 Nov. 2003, pp. 658-664.
2. B. Sheehan, *TICER: Realizable Reduction of Extracted RC Circuits*, proceedings of the IEEE International Conference on Computer-Aided Design, 7-11 Nov. 1999, pp. 200-203.
3. B. Sheehan, *Realizable Reduction of RC Networks,* IEEE Trans. Comput-Aided Design Integr. Circuits Syst., vol. 26, no. 8, pp. 1393-1407, Aug. 2007.

The primary advantages of reduction are:

- Results in smaller circuits, thus faster simulations.
- Requires less memory resources (uses less RAM and reduces swap space allocations).

Basic Reduction

The easiest way of activating the Eldo Reduction is by using the generic **.REDUCE ANALOG**=**YES|NO** command. The mandatory YES or NO value of the boolean ANALOG parameter selects some default settings usually suitable for analog or digital-like circuits respectively. The latter ones refer to circuits with square signals and for which the accuracy on the signals amplitude and delay are less stringent than for pure analog circuits. You may optionally specify a maximum delay error, noise error or cut-off frequency for the reduction process.

By default the Eldo generic **.REDUCE ANALOG**=**YES|NO** command automatically activates a combination of reduction methods expected to be well suited for the circuit being reduced according to its topology. However, this automatic selection may induce some CPU and memory overhead and it is highly recommended that for large-sized backannotations the type of (parasitic) network to be reduced be specified using the **REDUCE_NETWORK_TYPE** option.

Advanced Reduction

In addition to the generic **.REDUCE ANALOG**=**YES|NO** command some specific reduction commands for RLC or RC (**.REDUCE TICER** command), resistive-only reduction (**.REDUCE RONLY** command) or coupling-capacitance reduction (**.REDUCE CC** command) are available. Together with the available reduction options they provide a full control of the reduction process.

Main Parameters Controlling the Reduction Process

There are three main reduction parameters controlling the reduction methods activated by a generic **.REDUCE ANALOG**=**YES|NO** command, or more advanced specific reduction commands, described in the dedicated sections of this chapter.

The **noise error** parameter is defined as the amount of change in noise amplitude relative to a full strength signal. The noise-error value is a ratio, a unitless floating number. The default value is 0.01 (corresponding to 1%) for the **.REDUCE ANALOG**=**YES** command and 0.05 (5%) for the **.REDUCE ANALOG**=**NO** command.

Hereafter default values for both **.REDUCE ANALOG**=**YES** and **.REDUCE ANALOG**=**NO** commands are denoted *analog* default values and *digital-like* default values respectively.

The **delay error parameter** refers to the timing delay threshold used for reduction. The delay error units are seconds. The *analog* default is 0.5e-12 (a half picosecond); the *digital-like* default is 1.0e-12 (one picosecond).

The **cut-off frequency** refers to the upper limit of the frequency band ranging from DC upwards over which the Eldo Reduction is requested to maintain accuracy of the simulation results computed using the reduced circuit, within the effective noise error and delay error margins, as compared to the simulation results that would be obtained with the original unreduced circuit. Refer to the TICER reduction command below for a more detailed explanation of this cut-off frequency parameter.

Since the members of this parameter triplet are interdependent at most two of them can be specified. Moreover, *some of them*—even if specified explicitly on the reduction command—*may be overridden* according to parameters specified on some possibly conflicting reduction commands, according to the minimum frequency required to resolved the circuit input signals and possibly to the frequency range implied by the required simulation analyses as well. When such a situation happens, Eldo issues a warning indicating the effective value of the reduction parameter being modified.

Reduction Output

Some statistics are printed to allow you to assess the efficiency of the reduction process using the default or specified parameter values (noise error, delay error and/or cut-off frequency). These statistics may change between two successive Eldo versions as a result of a permanent development process toward increasing reduction efficiency. Nevertheless the accuracy of the simulation results, relative to those of the original unreduced circuit, is preserved as long as the reduction cut-off frequency, noise error and delay error are set adequately.

Generic Reduction Command

.REDUCE ANALOG

This command actives a combination of TICER, RONLY and/or CC reduction methods expected to be well suited for the circuit being reduced.

Usage

```
.REDUCE ANALOG=YES|NO
+ [DELAY_ERROR=value] [NOISE_ERROR=value]
.REDUCE ANALOG=YES|NO
+ [FREQUENCY=value] [DELAY_ERROR=value]
.REDUCE ANALOG=YES|NO
+ [FREQUENCY=value] [NOISE_ERROR=value]
```

Description

This generic command activates the Eldo Reduction using *analog* or *digital-like* default values. This command combines the effects of the **.REDUCE TICER** or **.REDUCE RONLY** commands and possibly the **.REDUCE CC** command as well. The latter commands take precedence over the **.REDUCE ANALOG**=**YES**|**NO** command if they are specified in the netlist.

At most two parameter values among **DELAY_ERROR**, **NOISE_ERROR** and **FREQUENCY** values may be specified. If a value is not specified for either, the default value is used.

The default value for **FREQUENCY** is computed accounting for the specified or default values of **NOISE_ERROR** and/or **DELAY_ERROR**, accounting for the circuit input signals and for the setup of the analyses to be simulated as well. When a conflict arises as a result of the specified parameter values, it is tentatively resolved by overriding either parameter value and an informative warning is issued.

Specific TICER, RONLY or CC reductions may be performed according to the related specific **.REDUCE** commands present in the netlist.

Arguments

- **ANALOG**=**YES**|**NO**

  Mandatory parameter and value.

  If YES, this parameter indicates to use the *analog* default values for the noise error, delay error or cut-off frequency parameter when they are not specified. This is the default behavior.

  If NO, this parameter indicates to use the *digital-like* default values for the noise error, delay error or cut-off frequency parameter when they are not specified.

- **DELAY_ERROR**=*value*

  Optional parameter and value specifying the timing delay threshold used for reduction. The delay units are seconds. The *analog* default is 0.5e-12 (a half picosecond); the *digital-like* default is 1.0e-12 (one picosecond).

- **NOISE_ERROR**=*value*

  Optional parameter and value specifying the error threshold used for reduction. The noise error is defined as the amount of change in noise amplitude relative to a full strength signal. The *noise* value is a ratio, a unitless floating number. The *analog* default value is 0.01 (1%); the *digital-like* one is 0.05 (5%).

- **FREQUENCY**=*value*

  The optional frequency parameter controls which nodes in the circuit TICER can select for subsequent elimination; the tool selects nodes with time constants less than a fourth of the inverse frequency parameter.

  Setting the frequency parameter a higher value results in larger (more R, C and L elements) circuits having a wider bandwidth of accuracy.

  Setting the frequency parameter a lower value results in more compression and an earlier roll-off in accuracy.

Calculate the frequency parameter value using the following formula:

frequency_value = *tradeoff_value* / *transition_time_minimum*

where:

tradeoff_value is a number between 4 and 10. A larger trade-off value results in a higher frequency parameter value and, consequently, more accurate and less aggressive reduction.

transition_time_minimum is the shortest rise or fall time expected in the design. In general it can be estimated as a fifth of the design switching delay.

TICER reduction preserves the frequency response of the circuit from DC up to the specified frequency value. Consequently, the frequency parameter controls trading off compression with accuracy.

Example

```
.REDUCE ANALOG=YES
```

Specifies that reduction will be done with the *analog* default values for **DELAY_ERROR** (0.5 ps) and **NOISE_ERROR** (1%).

```
.REDUCE ANALOG=NO
```

Specifies that reduction will be done with the *digital-like* default values for **DELAY_ERROR** (1 ps) and **NOISE_ERROR** (5%).

```
.REDUCE ANALOG=YES DELAY_ERROR=5e-12 NOISE_ERROR=0.15
```

Specifies a reduction with a delay error of 5 picoseconds and a noise error of 15%.

```
.REDUCE ANALOG=YES DELAY_ERROR=5e-12
```

Shows how one of the optional parameters may be used. In this case the *analog* default value for **NOISE_ERROR** (1%) will be used together with the specified delay error.

```
.REDUCE ANALOG=NO NOISE_ERROR=0.1
```

Shows how one of the optional parameters may be used. In this case the default *digital-like* value for **DELAY_ERROR** (1 ps) will be used together with the specified noise error.

TICER Reduction Command

.REDUCE TICER

This command specifies to perform TICER reduction in a way that has little impact on circuit characteristics such as delay up to a specified frequency. By default TICER RLC reduction using branch merging or TICER RC reduction is activated according to the circuit topology. Either may be forced by specifying the REDUCE_NETWORK_TYPE option value to “RLC” or “RC” respectively.

Usage

```
.REDUCE TICER FREQUENCY=value [ PORTMERGE [=value ]]
+ [ ANALOG=YES|NO ]
+ [ DELAY_ERROR=value | NOISE_ERROR=value ]
```

Description

This command activates electrically-based reduction. The reduction compresses RLC or RC networks into equivalent ones although the topology may be entirely different.

Basic TICER does not move or alter ports (connections between RLC networks and the remaining elements of the circuit). Port merging enables TICER RLC reduction to achieve more reduction than it might otherwise by combining certain ports that normally would be kept separate. In practice, many RLC circuits—especially local networks—are “port bound” in the sense that they have relatively many ports and relatively few internal nodes. Often basic TICER cannot reduce these RLC networks much because it can only eliminate internal nodes. With port merging, TICER can eliminate ports as well by transferring the ports onto neighboring nodes. This gives TICER reduction greater flexibility and allows it to achieve more reduction.

Note

It is especially important when using **PORTMERGE** to choose a correct setting for the TICER **FREQUENCY** parameter. If there are too many low-value resistors in the netlist, omit **PORTMERGE** from the **.REDUCE TICER** command or use a higher TICER cut-off frequency parameter value.

Note

The default or specified noise-error and delay-error values are guess values that may be automatically modified to account for cut-off frequency and portmerge timing threshold values, that take precedence over them. In such a case a warning message is issued to inform you of the effective values of noise error and delay error used. These values are also printed at the beginning of the reduction statistics summary, along with the reduction cut-off frequency.

Arguments

- **FREQUENCY**=*value*

  The mandatory frequency parameter controls which nodes in the circuit TICER can select for subsequent elimination; the tool selects nodes with time constants less than a fourth of the inverse frequency parameter value.

  Setting the frequency parameter a higher value results in larger (more R, C and L elements) circuits having a wider bandwidth of accuracy.

  Setting the frequency parameter a lower value results in more compression and an earlier roll-off in accuracy.

Calculate the frequency parameter value using the following formula:

frequency_value = *tradeoff_value* / *transition_time_minimum*

where:

tradeoff_value is a number between 4 and 10. A larger trade-off value results in a higher frequency parameter value and, consequently, more accurate and less aggressive reduction.

transition_time_minimum is the shortest rise or fall time expected in the design. In general it can be estimated as a fifth of the design switching delay.

TICER reduction preserves the frequency response of the circuit from DC up to the specified cut-off frequency value. Consequently, the frequency parameter controls trading off compression with accuracy.

- **PORTMERGE** [=*value*]

  An optional keyword that enables a more aggressive form of TICER reduction. This form combines ports whenever the change in timing is less than the specified value. The default timing tolerance is such that the impact on the time constants of ports is equivalent to that set by the reduction cut-off frequency for nodes. The default timing threshold value uses the following formula:

  $value = 1 / (8 \times frequency)$

  This parameter has no effect on TICER reduction alone when option **REDUCE_NETWORK_TYPE**=**RC** is set.

- **ANALOG**=**YES**|**NO**

  Optional parameter and value.

  If YES, specifies to use the *analog* default values of delay error and noise error. This is the default.

  If NO, specifies to use the *digital-like* default values of delay error and noise error.

- **DELAY_ERROR**=*value*

  Optional parameter specifying the timing delay threshold used for reduction. The delay units are seconds. The *analog* default is 0.5e-12 (a half picosecond); the *digital-like* default is 1.0e-12 (one picosecond).

This parameter has no effect on TICER reduction alone when option **REDUCE_NEWTORK_TYPE**=**RC** is set.

- **NOISE_ERROR**=*value*

  Optional parameter specifying the error threshold used for reduction. The noise error is defined as the amount of change in noise amplitude relative to a full strength signal. The value for noise error is a ratio, a unitless floating number. The *analog* default value is 0.01 (1%); the *digital-like* one is 0.05 (5%).

  This parameter has no effect on TICER reduction alone when option **REDUCE_NEWTORK_TYPE**=**RC** is set.

Example

```
.REDUCE TICER FREQUENCY=5GHz
```

Specifies a basic TICER reduction (without aggressive port merging) preserving accuracy of the frequency response from DC up to 5 GHz, and not using any default value of noise error or delay error.

```
.REDUCE TICER FREQUENCY=1GHz PORTMERGE=1ps ANALOG=YES
```

Specifies a TICER reduction with a 1 gigahertz cut-off frequency and aggressive portmerge reduction using a timing threshold of 1 picosecond, and also using the *analog* default values of noise error and delay error.

Reduction Command for Resistive Networks

.REDUCE RONLY

This command reduces resistive-only networks by eliminating all internal nodes.

Usage

```
.REDUCE RONLY [SPARSIFY = YES|NO]
```

Description

The **.REDUCE RONLY** command operates only on resistive networks, ones that do not include capacitance or inductance. It is most effective on networks with few ports and many internal nodes.

The R-Only operation reduces resistive networks so that they are electrically equivalent to the original ones, but the topology may be entirely different. The effective resistance between ports is preserved.

With the **SPARSIFY** optional parameter is set to YES, the circuit is further reduced by eliminating some resistors and adjusting the values of others to compensate.

The R-Only reduction is more aggressive than **.REDUCE ANALOG**=**YES**|**NO**, making it especially useful for large resistive networks, such as power distribution networks, that might otherwise exceed available computer resources. When both reduction commands are specified in the netlist, **.REDUCE RONLY** is applied to pure resistive networks and **.REDUCE TICER** / **.REDUCE CC** to the other networks.

Arguments

- **SPARSIFY**=**YES**|**NO**

  Optional keyword and value.

  If YES, directs that the resistive networks should be further reduced by combining resistors.

  If NO, this resistor combining reduction will not be performed. This is the default behavior.

Example

```
.REDUCE RONLY
```

Specifies that reduction will be done on a resistive only network.

Reduction Command for Coupled Capacitances

.REDUCE CC

This reduction command controls reduction of coupled capacitance between RLC or RC nets based on total capacitance.

Usage

```
.REDUCE CC
+ ABSOLUTE=value | RATIO=value | ABSOLUTE=value RATIO=value
+ [SCALE=value]
```

Description

This reduction command controls reduction of coupled capacitance between RLC or RC nets based on total capacitance.

When both an **ABSOLUTE** threshold value and a relative threshold **RATIO** value are specified, both conditions must be true before the coupled capacitance is grounded.

Note

The CC reduction occurs before the TICER reduction implied by a **.REDUCE ANALOG**=**NO** command, and after the TICER reduction implied by a **.REDUCE ANALOG**=**YES** command.

Arguments

- **ABSOLUTE**=*value*

  Keyword and absolute threshold value. If the total coupling capacitance between a pair of RLC or RC nets is less than the specified threshold value, that coupling is applied as grounded capacitances to both nets.

- **RATIO**=*value*

  Keyword and relative threshold value as a number between 0 and 1. The total coupled capacitance between two RLC or RC nets is compared to the total capacitance of each individually. If for both nets the coupled capacitance accounts for less than the relative threshold ratio (expressed as a decimal) of the total net capacitance, that coupling is applied as grounded capacitances for both nets.

- **SCALE**=*value*

  Optional keyword and value that applies the scale factor to the reduce capacitances before grounding. Only the coupled capacitances that are converted to grounded capacitance are scaled; the other capacitances are not modified. By default, the scale value is 1.0.

Example

The RATIO decoupling condition is met when the total coupling capacitance between two nets is less than the specified percent of both. For example, net A and net B are coupled by 26 fF. The total capacitance (intrinsic and coupled) on net A is 250 fF and on net B is 300 fF.

The following reduction command is used:

```
.REDUCE CC RATIO=0.1
```

Then the A-B coupling will not be grounded because although 26 fF is less than 10% of net B's 300 fF, it is greater than 10% of net A's 250 fF.

Reduction Options

Some options are provided with the aim of helping the reduction process to achieve greater efficiency or controlling the topological impact of reduction on the circuit.

Specifying the Network Type for Reduction

Option Syntax

```
.option REDUCE_NETWORK_TYPE = RLC | RC | RONLY
```

Description

This option specifies the type of networks to be reduced:

- **RLC** for (parasitic) networks containing resistors, capacitors and inductors. This is the default.
- **RC** for (parasitic) networks with only resistors and capacitors, no inductors.
- **RONLY** for pure resistive networks, no capacitors nor inductors.

Note

It is strongly recommended to set this option according to the parasitic network type for large-sized backannotations.

Note

This option may be used to control the reduction methods selected by the **.REDUCE ANALOG=YES|NO** command, by overriding the default network type selected according to the circuit topology.

Controlling the Effects of Reduction on Circuit Topology

Option Syntax

```
.option REDUCE_MAX_RES=value
.option REDUCE_MAX_CAP=value
.option REDUCE_MAX_IND=value
```

Description

This option specifies the maximum value of resistors, capacitors and inductors respectively, that are eligible for reduction. That is, elements with values greater than the specified value(s) will not be removed by reduction.

Note

These options may be used to avoid the removal of some resistors, capacitors and inductors that are not parasitics but are part of the active circuitry. The characteristics of some unstable circuits, such as oscillators, may indeed be affected by the removal of such elements.

Option Syntax

```
.option REDUCE_KEEP_OUTPUTS = YES|NO
```

Description

If **YES**, the default, this option ensures that no required output becomes unavailable as a result of reduction. If **NO**, it allows reduction to remove some nodes and/or Rs, Cs, Ls even if they are related to some requested outputs.

Note

Setting option **REDUCE_KEEP_OUTPUTS=NO** provides a convenient way for assessing whether reduction is hampered by a large number of requested outputs. In such a case, the options **REDUCE_KEEP_NODE** and **REDUCE_KEEP_INST** may be added to selectively recover the availability of some missing outputs.

Option Syntax

```
.option REDUCE_KEEP_NODE=name
```

Description

This option prevents the specified node from being eliminated by reduction. Double quotes are mandatory. By default, nodes related to output statements and nodes connected to devices other than a resistor, a capacitor or an inductor are not removed, unless option **REDUCE_KEEP_OUTPUTS=NO** is set.

Option Syntax

```
.option REDUCE_KEEP_INST=name
```

Description

This option prevents the specified resistor, capacitor or inductor instance from being eliminated by reduction. By default, instances related to output statements are not removed unless option **REDUCE_KEEP_OUTPUTS=NO** is set.

Chapter 16
Integral Equation Method (IEM)

Introduction

The Integral Equation Method (IEM) is a simulation algorithm developed for transient analog circuit simulation. With some circuits, IEM provides better performance than classic algorithms based on the Newton-Raphson method combined with a multi-step discretization scheme (for example Backward Euler, trapezoidal, gear, and so on).

In some cases, IEM may perform better than Newton methods in terms of stability and accuracy, which has a favorable incidence on speed. The improvements appear particularly for circuits requiring high precision or those which exhibit tight coupling or stability problems.

IEM's performance enhancement derives from the use of a semi-analytic approach combined with efficient numerical techniques.

Application Domains

IEM is invoked for TRANSIENT analysis of ANALOG circuits only. DC and AC analyses can only be performed using Newton-Raphson methods.

In case Eldo works together with another simulation kernel, digital or analog, then IEM is not supported.

Circuit Elements Supported—Limitations

As a rule, all circuit elements accepted by Eldo are supported by IEM except objects described by HDL-A, FAS or GUDM.

To be more explicit, elements actually supported by IEM are listed below.

- MOS models

Table 16-1. IEM Supported MOS Models

MERCK2	MERCK3	MERCK4	BERK1	BERK2	BERK3
BSIM1	BSIM2	STMOS1	STMOS3	EKV	MISNAN
BSIM3	CSEM	BSIM3V3	MOTOROLA	MOSP9	

- BJT models

Table 16-2. IEM Supported BJT Models

BERKBIP	STBIP				

- Diodes

Table 16-3. IEM Supported Diode Models

BERKDIO	ELDIO2	TUNNELD	STDIO	STFOWL	STMIS
JUNCAP					

- All other electrical device models, except:
  - o R, L, and C defined by the **`VALUE`** statement
  - o S and Z domain filters
- All independent or controlled sources are supported, except:
  - o Controlled sources defined by either the **`VALUE`** or **`TABLE`** statements (Note: Linear and polynomial controlled sources *are* supported)
- All digital and mixed-signal macromodels are supported
- Only comparator and delay elements are supported among analog macromodels
- Only the switch element is supported among switched capacitor macromodels
- Accusim magnetic models are supported but not those of Eldo.

In case a circuit or block contains an element not supported by IEM, this circuit or block is simulated using Newton-Raphson and a warning message is issued.

Use of IEM, Tolerance Parameters

The following simulation algorithms may be used on a circuit depending on the setting of the **`.OPTION`** line in the netlist and/or on the use of the (**`ANALOG`**) attribute on subcircuits:

- **`.OPTION NEWTON`**

  In this case the whole circuit is simulated in a single block, using the Newton method. Default.

- **`.OPTION IEM`**

  In this case the whole circuit is simulated in a single block, using IEM.

- **`.OPTION BLOCKS`**`=NEWTON`

  In this case, the circuit is partitioned (whenever possible) into one or more blocks. Simulation inside each block is performed using the Newton method and relaxation is undertaken between blocks using OSR. OSR is also used to simulate the non-Newton part of the circuit.

- **.OPTION BLOCKS=**IEM

  In this case, the circuit is partitioned (whenever possible) into one or more blocks. Simulation inside each block is performed using IEM and relaxation is undertaken between blocks using OSR. OSR is also used to simulate the non-IEM part of the circuit.

Note

The default simulation option is Newton, i.e. **.OPTION**=NEWTON

If a block contains either a GUDM, HDL-A or FAS model, then simulation inside that particular block will be performed using the Newton method. If no IEM blocks are created (because all blocks contain unsupported objects or the circuit is loosely coupled and can be simulated with OSR) then the IEM option is removed and a warning message is issued.

RELTOL and **VNTOL** parameters can be used in the same way as for SPICE-like simulators. These control both convergence of iteration loops *and* step size.

Note

For Newton-Raphson, **RELTOL** does *not* control the step size.

If **RELTOL** or **VNTOL** are not explicitly specified in the netlist, then **EPS** can be used to control their values, as follows:

- **RELTOL** = $(1.0\times10^{-9} \times \textbf{EPS})^{1/4}$
  RELTOL is then limited in the range 1.0×10^{-2} to 1.0×10^{-5}
- **VNTOL** = $(1.0\times10^{-7} \times \textbf{EPS})^{1/2}$
  VNTOL is then limited in the range 1.0×10^{-3} to 1.0×10^{-9}

For IEM, the option **LVLTIM** is by default set to 2, i.e. truncation error is used to control the time step. It is possible to set **LVLTIM=3** in the netlist, which will also allow the truncation error to control the time step. Additionally, it will impose calculated points at **TPRINT** values in the **.TRAN** command. Other settings of **LVLTIM** are not allowed since they produce less accurate results which would be meaningless with IEM.

All other SPICE compatible options (for example **ABSTOL**, **ITOL**, and so on) behave in the same way as for Newton-Raphson. This of course does not include integration scheme options (**BE**, **TRAP**, **GEAR**, ...) which do not apply to IEM.

In order to maintain compatibility with previous Eldo versions and Newton usage, **EPS** may still be used in the Eldo sense, i.e. as a means to control all precision parameters.

Efficient Usage of IEM

This section describes some details to help users achieve the best accuracy and performance results with IEM, used alone or in combination with OSR.

- IEM alone (**`.OPTION IEM`**)

  As already discussed above, this setup allows IEM to simulate the whole circuit in a single block. This is required for all-analog designs, or at least when the analog part of the circuit is much larger than the logic circuitry (i.e. approximately 75% of devices or more). This option is particularly recommended for circuits requiring high precision and/or those manifesting stability problems. IEM intrinsically gives very good accuracy and doesn't require tightening of accuracy parameters. Hence, the default **`EPS`** value is usually enough to achieve high quality simulation results. Lowering **`EPS`** will give even higher precision, but at the expense of CPU time.

- IEM + OSR (**`.OPTION BLOCKS=IEM`**)

  This setup allows Eldo to partition the circuit, assigning tightly coupled blocks into IEM, the remaining part being simulated with OSR. By default, MOS and grounded capacitors are assigned to OSR, hence *it is recommended to use this setup in conjunction with a correct imposition of the attribute (***`ANALOG`***) for relevant circuits.* This way, BJTs, resistors, and all MOS analog subcircuits will be simulated by IEM while other MOS blocks will be considered as logic ones and efficiently simulated by OSR (whenever possible).

  As usual, it is possible to increase accuracy by reducing **`EPS`**, however, as in all-IEM simulation this is not particularly recommended.

Note
The combination of IEM (a highly precise algorithm) with OSR (a fast but less precise algorithm) may not be justified in some cases.

Examples for IEM

Introduction

This section demonstrates the advantages of IEM, using the following set of examples:

- Stability
  - o *bjtinv_iem.cir* and *bjtinv_nrm.cir*
  - o *ivdd_iem.cir* and *ivdd_nrm.cir*

  Newton-Raphson (`nrm`) versions are included for comparison with the **iem.cir* files.

- Accuracy
  - o *oscil_iem.cir*, *oscil_nrm.cir* and *oscil_ref.cir*
  - o *moy1_iem.cir*, *moy1_nrm.cir* and *moy1_ref.cir*

  Newton-Raphson (**`nrm`**) and reference (**`ref`**) versions are included for comparison with the **iem.cir* files.
- Speed
  - o *ladder_iem.cir* and *ladder_nrm.cir*
  - o *opamp_5pin_iem.cir* and *opamp_5pin_nrm.cir*

  Newton-Raphson (**`nrm`**) versions are included for comparison with the **iem.cir* files.

The listings for these examples can be found in the following directory included with your software:

```
$MGC_AMS_HOME/examples/eldo
```

Note

For more examples of using Eldo please refer to the Examples appendix and the Tutorials chapter.

Stability

IEM is inherently stable as opposed to, for example, the Trapezoidal scheme which may sometimes exhibit numerical instabilities. These instabilities can be suppressed by reducing tolerances. Backward-Euler scheme also offers numerical damping but tolerances should be greatly decreased in order to obtain an acceptable precision. In both cases, this increases the CPU time. For IEM, these numerical instabilities do not appear even at default precision. The following examples (*bjtinv_iem.cir* and *ivdd_iem.cir*) demonstrate this effect.

Example 1—bjtinv

Complete Netlist

Note

The netlist is also provided as a Newton-Raphson version (*bjtinv_nrm.cir*) in the examples directory for comparison. The netlist is identical to *bjtinv_iem.cir* with the exception of **`.OPTION NEWTON`** selected as opposed to **`.OPTION IEM`**.

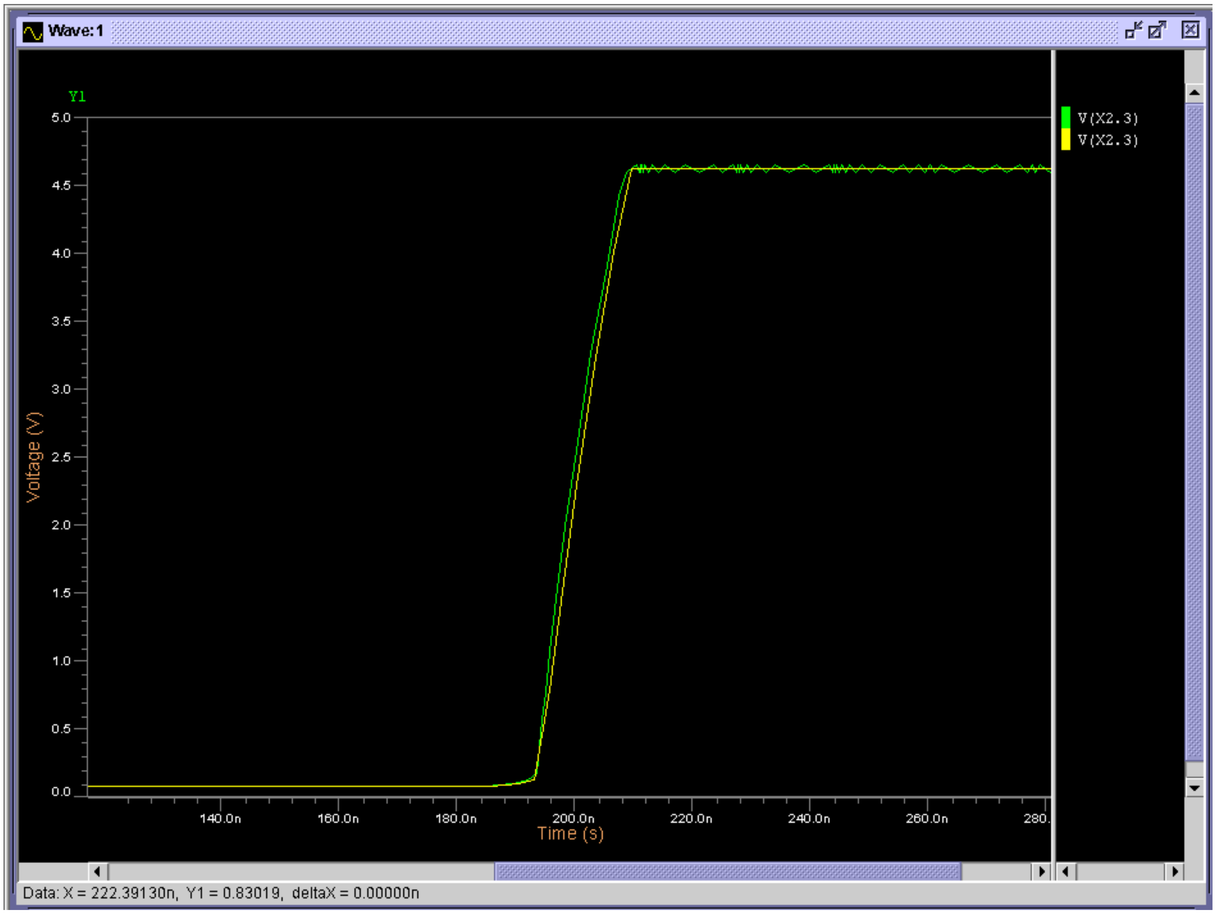
```
bjtinv_iem.cir
vcc 5 0 dc 5.
vin 1 0 pulse(0 5 10ns 40ns 40ns 110ns 300ns)
.subckt inv 5 1 6
```

```
q1 3 2 0 x33
q2 6 4 0 x33
rc1 5 3 1k
rc2 5 6 1k
rb1 1 2 10k
rb2 3 4 10k
.ends inv
x1 5 1 6   inv
x2 5 6 10  inv
x3 5 10 11 inv
x4 5 11 12 inv
x5 5 12 13 inv
x6 5 13 14 inv
.print tran v(1) v(11) v(14)
.plot  tran v(1) v(x2.3) v(11) v(14)
.tran 2ns 300ns
.model x33 npn(bf=80 rb=100 va=50 tf=.3e-9 tr=6.e-9)
.option iem accusim2
.end
```

Simulation Results

Figure 16-1 compares the waveform produced by the Newton-Raphson method (shown in green) compared with the waveform produced by the IEM method (shown in yellow).

Figure 16-1. Example 1—bjtinv



Example 2—ivdd

Note

Included in this directory are *ivdd_xxx.cir* examples which are intended for use with Eldo-ST. Eldo-ST is available for STMicroelectronics licensees only. Non-license holders may still simulate these examples but multiple warning messages will be generated.

Please refer to "How to Invoke Eldo-ST" on page 1509.

Complete Netlist

Note

The netlist is also provided as a Newton-Raphson version (*ivdd_nrm.cir*) in the examples directory for comparison. The netlist is identical to *ivdd_iem.cir* with the exception of **`.OPTION NEWTON`** selected as opposed to **`.OPTION IEM`**.

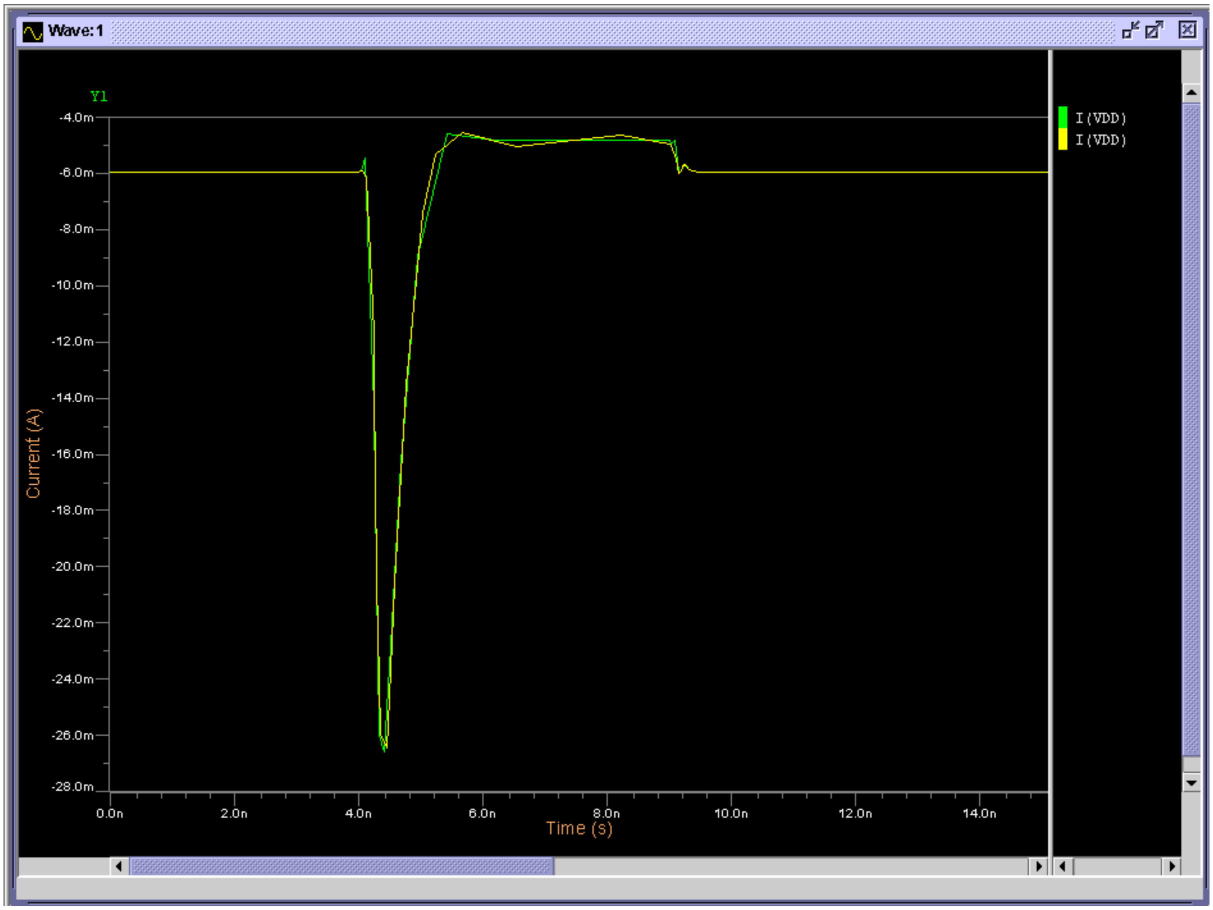
```
ivdd_iem.cir
.option XA= 1.500000e-06
*-------------------------------------------------------------* Main
Circuit Netlist:
*-------------------------------------------------------------C25 Z GND
8.5188e-16
C24 Z VDD 8.1332e-16
C23 N2 GND 6.3258e-15
C22 N2 VDD 6.79986e-15
C21 N2 Z 4.05478e-15
C20 A GND 9.072e-16
C19 A VDD 8.3745e-16
C18 A N2 8.05086e-16
M9 VDD A N2 VDD EP3 W=1.05571e-05 L=5e-07 AD=1.14175e-11
+ AS=1.1425e-11 PD=3.3e-06 PS=1.31e-05
M16 Z N2 VDD VDD EP3 W=1.07571e-05 L=5e-07 AD=5.22e-12
+ AS=1.0495e-11 PD=1.3e-06 PS=2.7e-06
M14 Z N2 VDD VDD EP3 W=1.07571e-05 L=5e-07 AD=5.22e-12
+ AS=1.0495e-11 PD=1.3e-06 PS=2.7e-06
M12 Z N2 VDD VDD EP3 W=1.07571e-05 L=5e-07 AD=5.22e-12
+ AS=1.0495e-11 PD=1.3e-06 PS=2.7e-06
M10 Z N2 VDD VDD EP3 W=1.07571e-05 L=5e-07 AD=5.22e-12
+ AS=1.14175e-11 PD=1.3e-06 PS=3.3e-06
M17 VDD N2 Z VDD EP3 W=1.07571e-05 L=5e-07 AD=1.7695e-11
+ AS=5.22e-12 PD=1.59e-05 PS=1.3e-06
M15 VDD N2 Z VDD EP3 W=1.07571e-05 L=5e-07 AD=1.0495e-11
+ AS=5.22e-12 PD=2.7e-06 PS=1.3e-06
M13 VDD N2 Z VDD EP3 W=1.07571e-05 L=5e-07 AD=1.0495e-11
+ AS=5.22e-12 PD=2.7e-06 PS=1.3e-06
M11 VDD N2 Z VDD EP3 W=1.07571e-05 L=5e-07 AD=1.0495e-11
+ AS=5.22e-12 PD=2.7e-06 PS=1.3e-06
M0 GND A N2 GND EN3 W=5.4e-06 L=5e-07 AD=5.9025e-12 AS=6e-12
+ PD=3.4e-06 PS=8.4e-06
M7 Z N2 GND GND EN3 W=5.25711e-06 L=5e-07 AD=2.87e-12
+ AS=5.445e-12 PD=1.3e-06 PS=3.2e-06
M5 Z N2 GND GND EN3 W=5.25711e-06 L=5e-07 AD=2.87e-12
+ AS=5.445e-12 PD=1.3e-06 PS=3.2e-06
M1 Z N2 GND GND EN3 W=5.25711e-06 L=5e-07 AD=2.87e-12
+ AS=5.9025e-12 PD=1.3e-06 PS=3.4e-06
M3 Z N2 GND GND EN3 W=5.25711e-06 L=5e-07 AD=2.87e-12
+ AS=5.445e-12 PD=1.3e-06 PS=3.2e-06
M8 GND N2 Z GND EN3 W=5.25711e-06 L=5e-07 AD=9.645e-12
+ AS=2.87e-12 PD=1.14e-05 PS=1.3e-06
M6 GND N2 Z GND EN3 W=5.25711e-06 L=5e-07 AD=5.445e-12
+ AS=2.87e-12 PD=3.2e-06 PS=1.3e-06
M4 GND N2 Z GND EN3 W=5.25711e-06 L=5e-07 AD=5.445e-12
+ AS=2.87e-12 PD=3.2e-06 PS=1.3e-06
M2 GND N2 Z GND EN3 W=5.25711e-06 L=5e-07 AD=5.445e-12
+ AS=2.87e-12 PD=3.2e-06 PS=1.3e-06
*************************************************************
.model en3
+ nmos            level=3
+ dmut =  1.474   dvt =-0.0011
+ eox =   11.3n   dl =  0.130u    dw = -0.100u
+ vt0 =  0.540    kb1 =  0.624    kb2 =  0.690
+ phi1 =  0.831   vc =  1.221     k0 =0.654E-04
+ tg =  0.064     racc =  411.2u  tw = -0.001u
+ a2 = -0.0074u   a3 = -0.076u    a1 = -0.260u
```

```
+ b2 =  0.004u      b3 =  0.032u      b1 =  0.055u
+ del=  0.294       gl = -0.255u      gw =  0.002u
+ delvg= -0.040     glvg =  0.152u    gwvg =  0.001u
+ ke =0.250E+08     l0 =  1.977u      eps =  0.018u
+ n1 = 0.330        n2 = 1.233        an =0.506E+06
+ bn =0.455E+08     cjs=8.02e-04      mjs=0.35
+ cjp=2.90e-10      mjp=0.09          cjc=1.37e-10
+ rec=0.04u         recl=0.1u         is=7.05e-04
+ ip=4.3e-09        xti=3.0           nd=2.0
.model ep3
+ pmos              level=3
+ dmut =  1.330     dvt =-0.0010
+ eox =   11.3n     dl =  0.140u      dw = -0.10u
+ vt0 =  0.600      kb1 =  0.680      kb2 =  0.680
+ phi1 =  0.829     vc =  0.000       k0 =0.152E-04
+ tg =  0.122       racc = 1071.2u    tw = -0.002u
+ a2 = -0.036u      a3 = -0.154u      a1 = -0.154u
+ b2 =  0.009u      b3 =  0.036u      b1 =  0.036u
+ del=  0.311       gl = -0.191u      gw =  0.010u
+ delvg= -0.036     glvg =  0.040u    gwvg = -0.011u
+ ke =0.370E+08     l0 =  0.574u      eps =  0.052u
+ n1 = 0.432        n2 = 1.069        an =0.219E+06
+ bn =0.734E+08     cjs=7.93e-04      mjs=0.33
+ cjp=2.29e-10      mjp=0.14          cjc=1.1e-10
+ rec=0.04u         recl=0.1u         is=8.00e-04
+ ip=4.0e-09        xti=3.0           nd=2.0
*******************************************
Vdd vdd 0 dc 3.6
Vin A 0 pwl (0 0 4n 0 4.25n 3.6 9n 3.6 9.25n 0)
*******************************************
Vdum Z Z1 dc 0
*******************************************
Cload Z1 0 3.2p
*******************************************
.temp 25
.tran 0.005n 32n
.connect 0 GND
.option accusim2
.option iem  !stable
*.option newton !trapezoidal exhibits numerical oscillations
.plot tran i(vdd)
.plot tran i(vdum)
.defwave wivdd=abs(i(vdd))
.defwave wivdum=abs(i(vdum))
.extract integ(w(wivdd))
.extract integ(w(wivdum))
.end
```

Simulation Results

Figure 16-2 compares the waveform produced by the Newton-Raphson method (shown in yellow) compared with the waveform produced by the IEM method (shown in green).

Figure 16-2. Example 2—ivdd



Accuracy

The precision is measured by comparing the results for each algorithm to a reference. The reference can be obtained using either of the two algorithms, by setting extremely low values on tolerance parameters. By default, IEM gives results at relatively high precision. In order to attain the same accuracy using Newton-Raphson, tolerance parameters should be tightened, which results in an increase in the CPU. The performance gain is measured by the CPU ratio between both algorithms when the output results are at the same level of accuracy.

Example 3—oscil

For this oscillator, a jitter lower than 2%, can be obtained by IEM at default settings, or by Newton-Raphson at **`EPS=3.0e-7`**. At this level of accuracy, the speed-gain for IEM is about 3×.

Complete Netlist

Note

The netlist file is also provided as Newton-Raphson and reference versions (*oscil_nrm.cir* and *oscil_ref.cir* respectively) in the examples directory. *oscil_ref.cir* is the reference model to which the `iem` and `nrm` circuits are compared. The netlists are identical to *oscil_iem.cir* with the exception of **`.OPTION NEWTON`** selected for *oscil_nrm.cir* and **`.OPTION NEWTON EPS=1.e-8`** selected for *oscil_ref.cir*.

```
oscil_iem.cir
.MODEL PBSIM2 PMOS  level=11
+ TNOM=2.70000E+01  RSH=6.50000E+01   TOX=1.50000E-02
+ VDD=5.00000E+00   VGG=5.00000E+00   VBB=-5.00000E+00
+ JS=2.00000E-06    XPART=0.00000E+00 CGSO=2.00000E-10
+ CGDO=2.00000E-10  CGBO=0.00000E+00  PB=6.75000E-01
+ MJ=4.30000E-01    PBSW=0.10000E+00  MJSW=4.30000E-01
+ CJ=3.80000E-04    CJSW=1.20000E-10  DL=2.75240E-01
+ DW=2.15737E-01    MU0=4.88029E+02   N0=1.40000E+00
+ LN0=0.00000E+00   WN0=0.00000E+00   NB=5.00000E-01
+ LNB=0.00000E+00   WNB=0.00000E+00   ND=0.00000E+00
+ LND=0.00000E+00   WND=0.00000E+00   PHI=7.71089E-01
+ LPHI=1.66784E+00  WPHI=-3.12807E-01 MU0B=0.00000E+00
+ LMU0B=0.00000E+00 WMU0B=0.00000E+00 K1=8.10175E-01
+ LK1=0.00000E+00   WK1=0.00000E+00   K2=4.67768E-02
+ LK2=0.00000E+00   WK2=0.00000E+00   ETA0=-1.78327E-02
+ LETA0=2.74604E-02 WETA0=0.00000E+00 U10=-1.46747E-02
+ LU10=2.66063E-01  WU10=0.00000E+00  VFB=-9.74625E-01
+ LVFB=-2.01651E+00 WVFB=4.68098E-01  ETAB=0.00000E+00
+ LETAB=0.00000E+00 WETAB=0.00000E+00 U1B=0.00000E+00
+ LU1B=0.00000E+00  WU1B=0.00000E+00  MU20=1.32844E+00
+ LMU20=1.75239E+00 WMU20=0.00000E+00 MU2B=0.00000E+00
+ LMU2B=0.00000E+00 WMU2B=0.00000E+00 MU30=1.29954E+00
+ LMU30=-9.23397E+0 WMU30=0.00000E+00 MUS0=5.30129E+02
+ LMUS0=9.17775E+00 WMUS0=0.00000E+00 MUSB=0.00000E+00
+ LMUSB=0.00000E+00 WMUSB=0.00000E+00 MU2G=-4.77583E-01
+ LMU2G=0.00000E+00 WMU2G=0.00000E+00 MU3B=0.00000E+00
+ LMU3B=0.00000E+00 WMU3B=0.00000E+00 MU3G=1.17696E+00
+ LMU3G=0.00000E+00 WMU3G=0.00000E+00 MU40=-8.24961E-01
+ LMU40=7.86998E-01 WMU40=0.00000E+00 MU4B=0.00000E+00
+ LMU4B=0.00000E+00 WMU4B=0.00000E+00 MU4G=-1.19857E-01
+ LMU4G=0.00000E+00 WMU4G=0.00000E+00 UA0=2.89822E-02
+ LUA0=2.68543E-01  WUA0=-5.76159E-02 UAB=-1.12017E-02
+ LUAB=0.00000E+00  WUAB=0.00000E+00  UB0=1.22936E-02
+ LUB0=-1.08792E-02 WUB0=1.58744E-03  UBB=2.34752E-04
+ LUBB=0.00000E+00  WUBB=0.00000E+00  U1D=0.00000E+00
+ LU1D=0.00000E+00  WU1D=0.00000E+00  VGHIGH=2.00000E-01
+ LVGHIGH=0.000E+00 WVGHIGH=0.000E+00 VOF0=1.80000E+00
+ LVOF0=0.00000E+00 WVOF0=0.00000E+00 VOFB=0.00000E+00
+ LVOFB=0.00000E+00 WVOFB=0.00000E+00 VOFD=0.00000E+00
+ LVOFD=0.00000E+00 WVOFD=0.00000E+00 AIB=0.00000E+00
+ LAIB=0.00000E+00  WAIB=0.00000E+00  BIB=0.00000E+00
+ LBIB=0.00000E+00  WBIB=0.00000E+00  VGLOW=-1.50000E-01
+ LVGLOW=0.000E+00  WVGLOW=0.0000E+00 DELL=0.00000E+00
+ BEX=-1.50000E+00  TCV=1.00000E-03   AI0 =-2.70194E-02
+ LAI0= 1.03485E-01 WAI0= 0.00000E+00 BI0 = 6.95477E-01
```

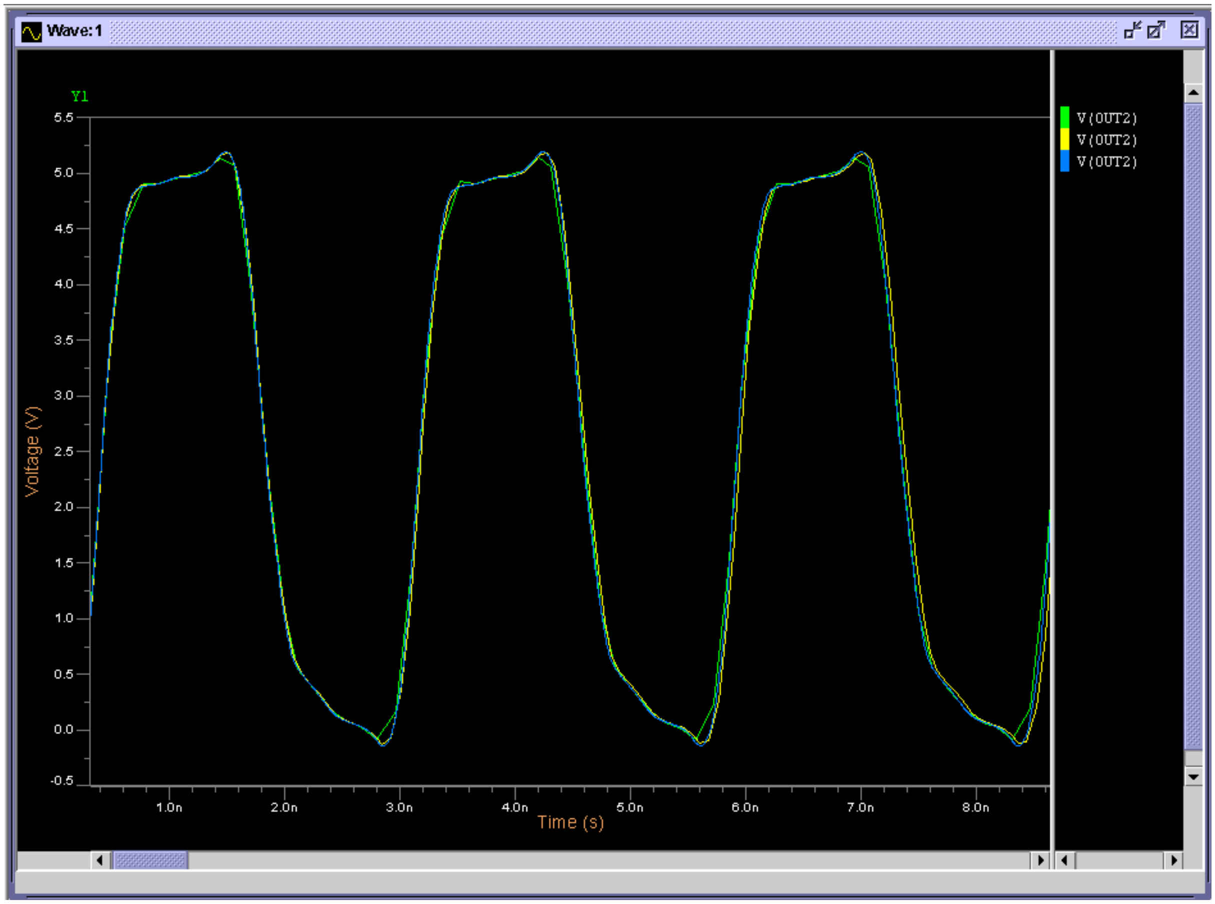
```
+ LBI0= 5.20197E-01 WBI0= 0.00000E+00
.MODEL NBSIM2 NMOS  level=11
+ TNOM=2.70000E+01  RSH=6.50000E+01   TOX=1.50000E-02
+ VDD=5.00000E+00   VGG=5.00000E+00   VBB=-5.00000E+00
+ JS=2.00000E-06    XPART=0.00000E+00 CGSO=2.00000E-10
+ CGDO=2.00000E-10  CGBO=0.00000E+00  PB=6.75000E-01
+ MJ=4.30000E-01    PBSW=0.10000E+00  MJSW=4.30000E-01
+ CJ=3.80000E-04    CJSW=1.20000E-10  DL=2.75240E-01
+ DW=2.15737E-01    MU0=4.88029E+02   N0=1.40000E+00
+ N0=0.00000E+00    WN0=0.00000E+00   NB=5.00000E-01
+ LNB=0.00000E+00   WNB=0.00000E+00   ND=0.00000E+00
+ LND=0.00000E+00   WND=0.00000E+00   PHI=7.71089E-01
+ LPHI=1.66784E+00  WPHI=-3.12807E-01 MU0B=0.00000E+00
+ LMU0B=0.00000E+00 WMU0B=0.00000E+00 K1=8.10175E-01
+ LK1=0.00000E+00   WK1=0.00000E+00   K2=4.67768E-02
+ LK2=0.00000E+00   WK2=0.00000E+00   ETA0=-1.78327E-02
+ LETA0=2.74604E-02 WETA0=0.00000E+00 U10=-1.46747E-02
+ LU10=2.66063E-01  WU10=0.00000E+00  VFB=-9.74625E-01
+ LVFB=-2.01651E+00 WVFB=4.68098E-01  ETAB=0.00000E+00
+ LETAB=0.00000E+00 WETAB=0.00000E+00 U1B=0.00000E+00
+ LU1B=0.00000E+00  WU1B=0.00000E+00  MU20=1.32844E+00
+ LMU20=1.75239E+00 WMU20=0.00000E+00 MU2B=0.00000E+00
+ LMU2B=0.00000E+00 WMU2B=0.00000E+00 MU30=1.29954E+00
+ LMU30=-9.23397E+00 WMU30=0.00000E+00 MUS0=5.30129E+02
+ LMUS0=9.17775E+00 WMUS0=0.00000E+00 MUSB=0.00000E+00
+ LMUSB=0.00000E+00 WMUSB=0.00000E+00 MU2G=-4.77583E-01
+ LMU2G=0.00000E+00 WMU2G=0.00000E+00 MU3B=0.00000E+00
+ LMU3B=0.00000E+00 WMU3B=0.00000E+00 MU3G=1.17696E+00
+ LMU3G=0.00000E+00 WMU3G=0.00000E+00 MU40=-8.24961E-01
+ LMU40=7.86998E-01 WMU40=0.00000E+00 MU4B=0.00000E+00
+ LMU4B=0.00000E+00 WMU4B=0.00000E+00 MU4G=-1.19857E-01
+ LMU4G=0.00000E+00 WMU4G=0.00000E+00 UA0=2.89822E-02
+ LUA0=2.68543E-01  WUA0=-5.76159E-02 UAB=-1.12017E-02
+ LUAB=0.00000E+00  WUAB=0.00000E+00  UB0=1.22936E-02
+ LUB0=-1.08792E-02 WUB0=1.58744E-03  UBB=2.34752E-04
+ LUBB=0.00000E+00  WUBB=0.00000E+00  U1D=0.00000E+00
+ LU1D=0.00000E+00  WU1D=0.00000E+00  VGHIGH=2.00000E-01
+ LVGHIGH=0.000E+00 WVGHIGH=0.000E+00 VOF0=1.80000E+00
+ LVOF0=0.00000E+00 WVOF0=0.00000E+00 VOFB=0.00000E+00
+ LVOFB=0.00000E+00 WVOFB=0.00000E+00 VOFD=0.00000E+00
+ LVOFD=0.00000E+00 WVOFD=0.00000E+00 AIB=0.00000E+00
+ LAIB=0.00000E+00  WAIB=0.00000E+00  BIB=0.00000E+00
+ LBIB=0.00000E+00  WBIB=0.00000E+00  VGLOW=-1.50000E-01
+ LVGLOW=0.0000E+00 WVGLOW=0.0000E+00 DELL=0.00000E+00
+ BEX=-1.50000E+00  TCV=1.00000E-03   AI0=-2.70194E-02
+ LAI0= 1.03485E-01 WAI0 = 0.0000E+00 BI0= 6.95477E-01
+ LBI0= 5.20197E-01 WBI0 = 0.0000E+00
Vdd vdd 0 dc 5
Vss vss 0 dc 0

.subckt inv in out vss vdd
r1 vdd vdd1 10
r2 vss vss1 10
M1 out in vss1 vss nbsim2 L=2U W=6U AD=90E-12 AS=90E-12 PD=24u
+ PS=24u
M2 out in vdd1 vdd pbsim2 L=2U W=10U AD=90E-12 AS=90E-12
+ PD=24u PS=24u
.ends inv
```

```
Xin1 in1 out1 vss vdd inv
c1 out1 0 0.1f
Xin2 out1 out2 vss vdd inv
c5 out2 0 0.1f
Xin3 out2 out3 vss vdd inv
c2 out3 0 0.1f
Xin4 out3 out4 vss vdd inv
c3 out4 0 0.1f
Xin5 out4 in1 vss vdd inv
c4 in1 0 0.1f

.tran 1n 100n uic

.ic v(out1)=5 v(out2)=0 v(out3)=5
.plot tran v(out1)
.plot tran v(out2)
.plot tran v(out3)
.plot tran v(out4)
.option accusim2
.option iem
.end
```

Simulation Results

Figure 16-3 compares the waveform produced by the Newton-Raphson method (shown in yellow) compared with the waveform produced by the IEM method (shown in green) and the waveform produced by the reference model (shown in blue).

Figure 16-3. Example 3—oscil



Example 4—moy1

The matching considerations in transistor current sources are showed in this circuit. By default, IEM gives an error in the average current difference of 32 nA. The same precision is attained by Newton-Raphson at **EPS**=1.0e-6. At this level the speed-gain for IEM is about 2×.

Complete Netlist

Note

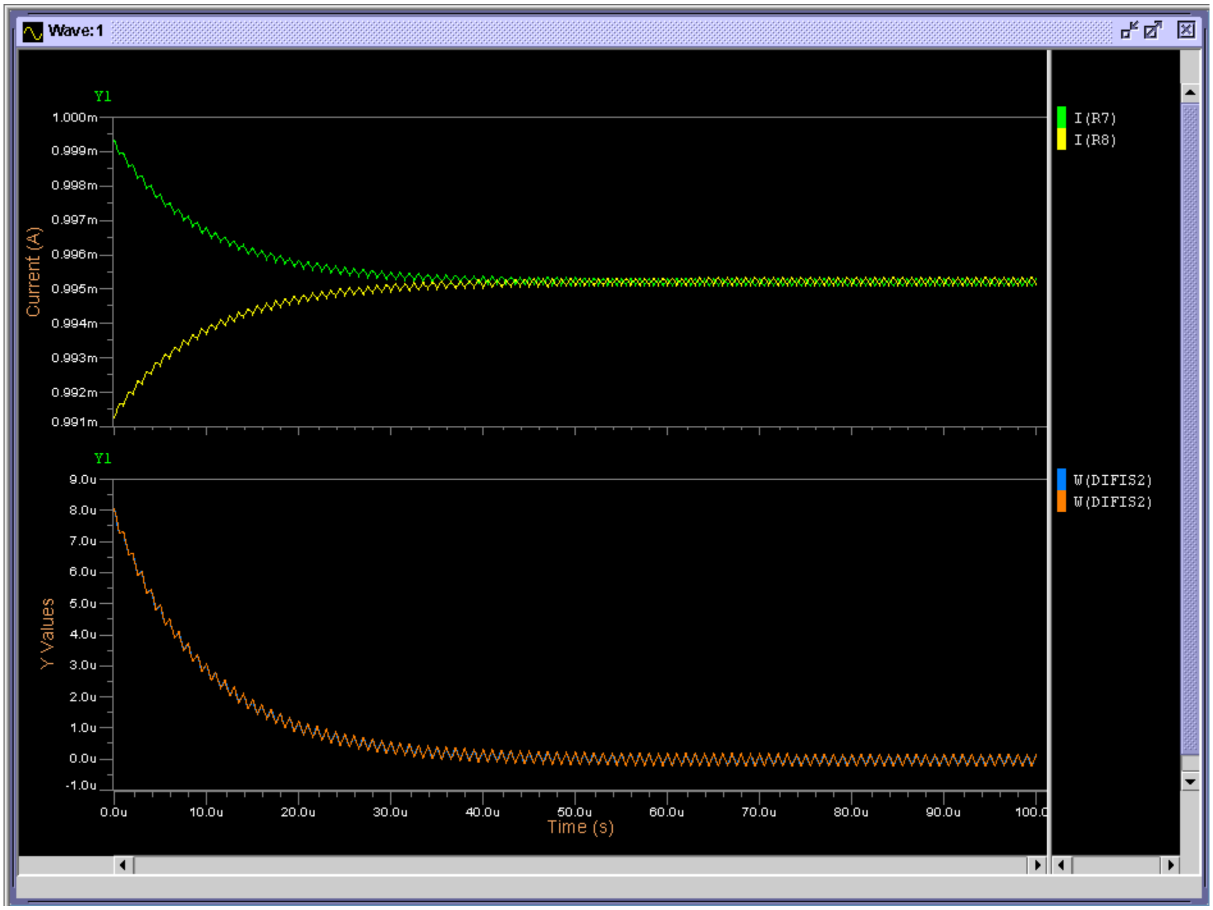
The netlist file is also provided as Newton-Raphson and reference versions (*moy1_nrm.cir* and *moy1_ref.cir* respectively) in the examples directory. *moy1_ref.cir* is the reference model to which the `iem` and `nrm` circuits are compared. The netlists are identical to *moy1_iem.cir* with the exception of **.OPTION NEWTON** selected for *moy1_nrm.cir* and **.OPTION NEWTON EPS**=1.e-8 selected for *moy1_ref.cir*.

```
moy1_iem.cir
.MODEL MN NMOS LEVEL= MERCK2 EOX= 200E-10 MU0= 500
+ DPHIF= 0.8 DW= 1.0E-6 DL= 0.1E-6 VT0= 0.75
+ KB= 0.6 REC= 0.1E-6 TG= 0.08 VL= 1.0E5 GL= 0.5E-6
+ KL= 0.3E-6 KW= 0.2E-6 DINF= 0.3 GW= 0.4E-6
+ LDIF= 3.5E-6 CDIFS0= 1.4E-4 CDIFP0= 8E-10
+ VE= 20E4 LMIN= 1.2E-6 WMIN= 3.4E-6 RSH= 500

mr   vp vp   r   0 mn w=200u l=2u
rr    r  0   1k
m1    3 vp   1   0 mn w=200u l=2u
r1    1  0   1k
m2    4 vp   2   0 mn w=200u l=2u
r2    2  0   1.01k
m3    5 hb   3   0 mn w=200u l=2u
m4    6 hb   4   0 mn w=200u l=2u
m5    6  h   3   0 mn w=200u l=2u
m6    5  h   4   0 mn w=200u l=2u
m7    7  a   5   0 mn w=200u l=2u
m8    8  a   6   0 mn w=200u l=2u
r7   s1  7   1k
c7    7  0   10n
r8   s2  8   1k
c8    8  0   10n
* alims
vs1 s1   0 dc 10
vs2 s2   0 dc 10
va    a  0 dc 7
ipol 0 vp   1m
.param d=2n
.param thold=490n+d
vh    h  0 pulse( 3   4 10n 10n 10n thold 1u)
vhb hb   0 pulse( 4   3 10n 10n 10n thold 1u)
* simuls
.dc
.op
.tran 100n 100u
.defwave difis2=i(r7)-i(r8)
.plot tran i(r7) i(r8)
.plot tran w(difis2)
.width out=80
.option accusim2
.option iem
*.option newton eps=1.e-6
.option noascii
.extract average(w(difis2),98u,100u)
.end
```

Simulation Results

Figure 16-4 compares the waveform produced by the Newton-Raphson method (shown in green) compared with the waveform produced by the IEM method (shown in yellow).

Figure 16-4. Example 4—moy1



Speed

Speed gain ratio depends on the nature of the circuit simulated as well as the level of activity. Slowly varying circuits may manifest no gain. In the following circuits, a certain level of activity was present such as to show IEM advantage at default settings for both IEM and Newton-Raphson.

Example 5—ladder

The speed-gain for IEM is about 2.3× (as compared to Newton-Raphson).

Complete Netlist

Note

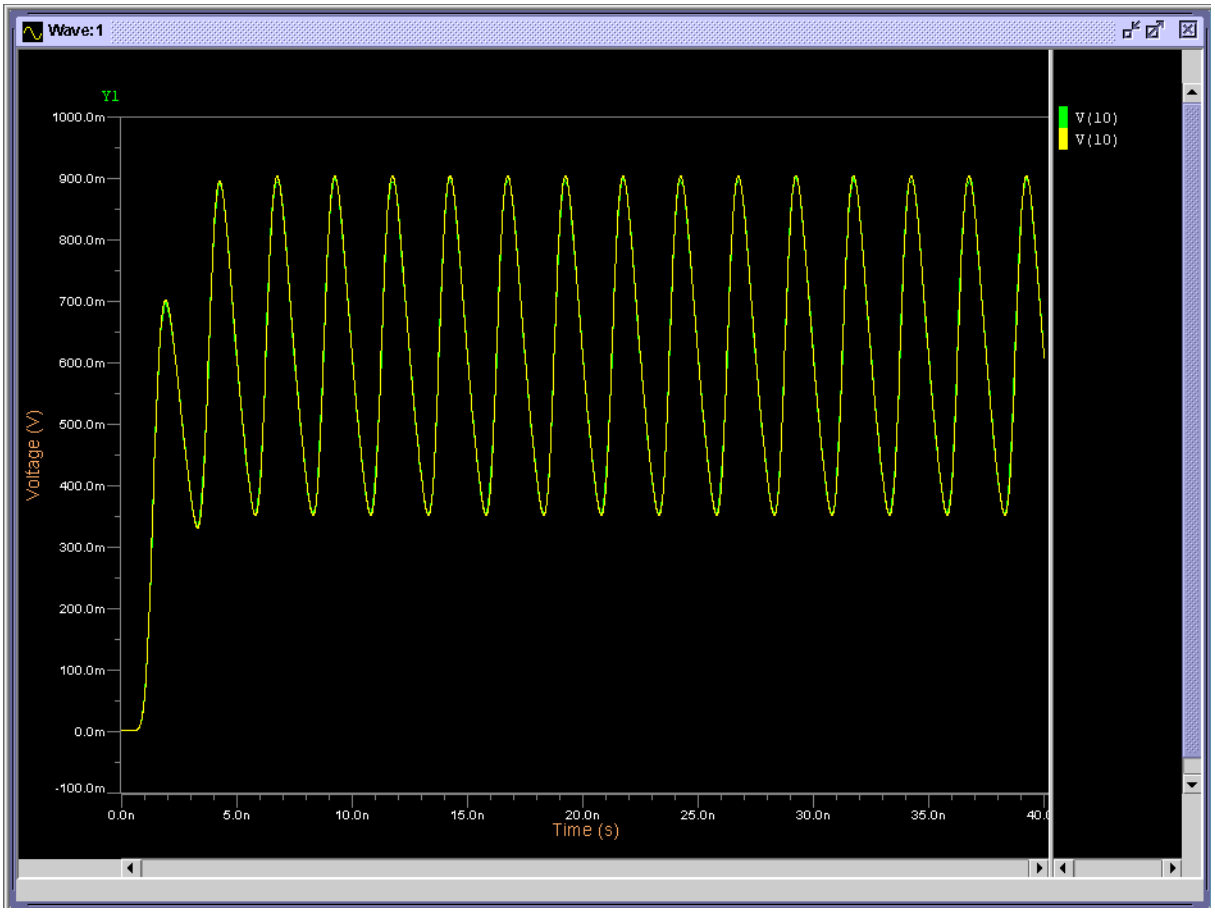
The netlist file is also provided as a Newton-Raphson version (*ladder_nrm.cir*) in the examples directory for comparison. The netlist is identical to *ladder_iem.cir* with the exception of **`.OPTION NEWTON`** selected for *ladder_nrm.cir*.

```
Simple R-Ladder Circuit
.param r=1K
.subckt r2r 1 2
R1 1 2 r
R2 2 0 {2*r}
D 0 2 DMOD OFF
.MODEL DMOD D CJO=1P
.ends r2r
*VIN 1 0 pulse 0 1024 0 1n 1n 100U 1
vin 1 0 sin 1 1024 0.4g
x1 1 2 r2r
x2 2 3 r2r
x3 3 4 r2r
x4 4 5 r2r
x5 5 6 r2r
x6 6 7 r2r
x7 7 8 r2r
x8 8 9 r2r
R1 9 10 r
ROUT 10 0 r
.width out=80
.tran 0.1N 40N
.plot tran v(10)
.option accusim2
.option iem
*.option newton
.end
```

Simulation Results

Figure 16-5 compares the waveform produced by the Newton-Raphson method (shown in green) compared with the waveform produced by the IEM method (shown in yellow).

Figure 16-5. Example 5—ladder



Example 6—opamp_5pin

For this circuit, the speed-gain for IEM is about 2.3× (as compared to Newton-Raphson).

Complete Netlist

Note

The netlist file is also provided as a Newton-Raphson version (*opamp_5pin_nrm.cir*) in the examples directory for comparison. The netlist is identical to *opamp_5pin_iem.cir* with the exception of **.OPTION NEWTON** selected for *opamp_5pin_nrm.cir*.

```
opamp_5pin_iem.cir
.option TNOM=27
VIZ11 4 0 10V
VIZ9 0 5 10V
VDZ0 6 0 SIN 0 1 1000 0 0
*.dc vdz0 1 -1 -0.01
*.plot dc v(3)
RIZ16 2 3 10K
```

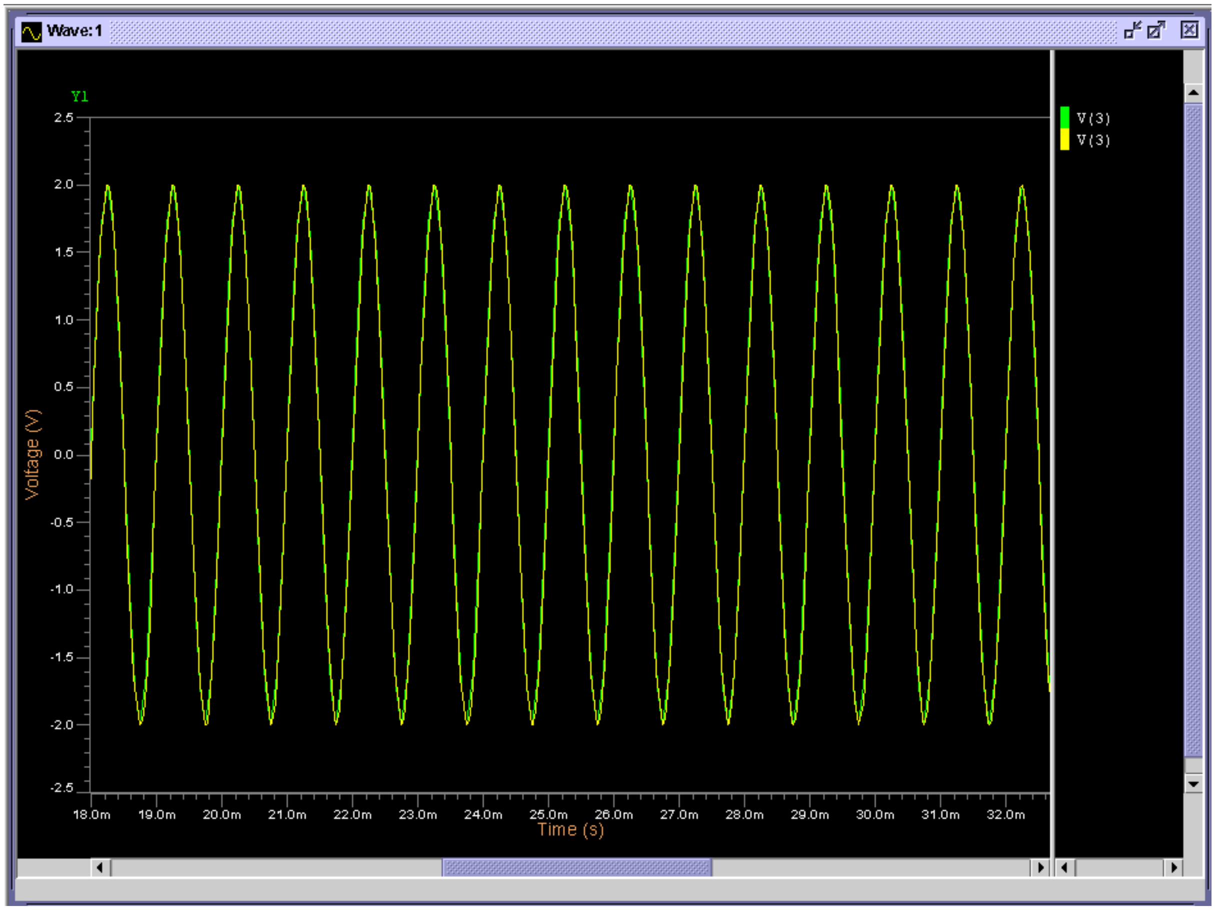
```
RIZ14 6 1 10K
RIZ13 6 0 10K
RIZ12 0 2 10K
RIZ3 3 0 10K
XA1 2 1 3 4 5  LM107
* OPAMP MACROMODEL $LM107
* NATIONAL  LINEAR P3-140, 1982
* SUPPLY VOLTAGE(S): 15
* USAGE: XNAME <-> <+> <OUT> <VCC> <VEE> $LM107
.SUBCKT LM107 2 3 6 7 4
V1 1004 0 1
DX1 0 1000 DMOD
DX2 0 1001 DMOD
DX3 1000 1002 DMOD
DX4 1001 1003 DMOD
VSINK 1002 1004 0
VSOURCE 1003 1004 0
FOUT 1000 1001 VO 1
FSOURCE 7 126 VSOURCE 1
FSINK 296 4 VSINK 1
IXX 6 0 0
VO 158 6 0
EVCC 126 5 7 5 1
EVEE 296 5 4 5 1
VLM2 5 115 55.7517
VLM1 116 5 55.7517
DD1 16 5 DD OFF
DD2 5 16 DD OFF
DCLP 158 112 DMOD OFF
DCLN 113 158 DMOD OFF
DLM1 106 116 DMOD OFF
DLM2 115 106 DMOD OFF
GDM 16 5 8 9 31.4165U
GCM 15 5 12 5 -21.2896M
GB 15 5 16 5 427.5729
JP 117 4 4 MMOD
EGND 5 4 7 4 0.5
ELIM 105 5 15 6 31.6228
VCHAIN 7 117 0
HCMR1 7 108 VCHAIN 222.535
HCMR2 110 4 VCHAIN 667.605
HCLP 126 112 VCHAIN 965.444
HCLN 113 296 VCHAIN 965.444
FEE 12 110 VCHAIN 1.4242M
FLIM 15 5 V40 1
V40 105 106 0
CE 12 5 1.25P
C1 8 9 2.5P
C2 15 16 5P
Q1 8 2 10 QM1
Q2 9 3 11 QM2
RE1 10 12 11.0128K
RC1 8 108 31.8304K
RE2 11 12 11.0128K
RC2 9 108 31.8304K
REE 12 5 1.0042G
*.opt abstol=1P vntol=1U reltol=1M
R2 5 16 0.1MEG
```

```
RO2 5 15 134
RO1 158 15 66
.MODEL QM1 NPN (IS=0.8F BF=40.6504)
.MODEL QM2 NPN (IS=0.8216F BF=42.735)
.MODEL DMOD D (N=1)
.MODEL DD D (N=30.1931)
.MODEL MMOD NJF (VTO=-0.6 BETA=4.993M)
.ENDS LM107
* BEGIN ELDO
.temp 27
.op

.TRAN 0.001 0.05 0
.plot tran v(3)
.plot tran v(6)
.width out=80
.option accusim2
.option iem
*.option newton
.end
```

Simulation Results

Figure 16-6 compares the waveform produced by the Newton-Raphson method (shown in green) compared with the waveform produced by the IEM method (shown in yellow).

Figure 16-6. Example 6—opamp_5pin



Chapter 17
Pole-Zero Post-Processor

Introduction

Eldo is able to obtain the locations of poles and zeros for analog circuits following a small signal analysis. This data is useful for obtaining stability and phase margin information as well as predicting closed loop performance from an open loop response. As a lot of closely spaced poles or zeros are not easily discernible via a Bode plot, the interactive post processor and its textual output clearly help to identify such singularities and evaluate the most important areas in a circuit's behavior.

The pole-zero data is obtained by linearizing the circuit about an operating point and outputting information about circuit matrix description and AC response. This data will be analyzed by the Eigenvalue QZ algorithm, one of the most accurate techniques available.

The pole-zero post-processor allows the designer to detect all the roots of the analyzed circuit and to simplify this information extracting the most meaningful of these roots. Moreover, a high level model of the reduced circuit—equivalent for AC and small signal analysis—is calculated and given in the form of an FNS device which can replace the original circuit when used as part of a more complex design.

The pole-zero post-processor may be activated for circuits which have been simulated using the **.PZ** and **.AC** commands. The **.PZ** command can take either the voltage at a node, between two nodes or the current through a voltage source as parameters. The pole-zero analysis determines the transfer function relationship—expressed as complex poles and zeros—between the input where the AC voltage source is applied and the output specified as a parameter in the **.PZ** command.

Pole-zero only works on quasi-static devices; pole-zero would give incorrect results on HDL-A models which are not linear with OMEGA (exp(OMEGA) for instance). However, for HDL-A models that make use of derivatives of order higher than 2 it is possible to declare additional IMPLICIT states and make the model linear in OMEGA.

Example

In ‘pseudo’ HDL-Av1 code, convert:

```
b = ddt(a)
c = ddt(b);
```

into:

```
EQUATION (b,c) FOR ac,dc,transient =>
b == ddt(a);
c == ddt(b);
```

Running the Dialog for Pole-Zero Analysis

Before running the pole-zero post-processor, you must make sure that an AC analysis has been performed on the circuit of interest and that a **`.PZ`** command has been included in the netlist. This has the effect of producing a file called <*circuit*>*.pz* in the output directory.

The pole-zero post-processor can be run in batch mode or interactive mode:

- Batch mode, run:

  pz <circuit>.pz -batch

  The pole-zero post-processor will run until completion using default answers to questions usually prompted by the tool

- Interactive mode, run:

  pz <circuit>.pz

  In this mode, you type in answers to all questions prompted by the tool. The arguments to provide are the following:

  `-th value`
  threshold for pole/zero cancellation

  `-cutoff value`
  cutoff frequency

  `-pindices indice_list -endpindices`
  list of indexes for pole selection; indices must be in increasing order

  `-zindices indice_list -endzindices`
  list of indexes for zero selection; indices must be in increasing order

  Note that if one of the arguments is provided, then answers to those pz questions which do not have explicit answers in the argument list will take the default answer, that is, the same as for the **-batch** mode.

The first result is the impression of all the roots found for the circuit, ordered by magnitude but with zeros divided in negative and positive real parts, as follows for a typical example:

```
Poles  Modulus       Real Part      Imaginary Part
1      6.264953E+00  -6.264953E+00   0.000000E+00
2      3.853866E+06  -3.824006E+06  -4.788095E+05
3      3.853866E+06  -3.824006E+06   4.788095E+05
4      4.396770E+06  -4.396770E+06   0.000000E+00
5      5.423444E+06  -5.423444E+06   0.000000E+00
6      6.554762E+06  -3.126871E+06  -5.760866E+06
7      6.554762E+06  -3.126871E+06   5.760866E+06
8      6.584083E+06  -5.150248E+06  -4.101840E+06
```

```
9      6.584083E+06  -5.150248E+06   4.101840E+06
10     1.343013E+07  -1.343013E+07   0.000000E+00
11     1.531050E+07  -1.531050E+07   0.000000E+00
12     2.217833E+07  -2.217833E+07   0.000000E+00
13     4.927351E+07  -4.927351E+07   0.000000E+00
14     6.564085E+07  -6.564085E+07   0.000000E+00
15     1.228260E+08  -1.228260E+08   0.000000E+00
Zeros  Modulus       Real Part      Imaginary Part
1      3.853866E+06  -3.824006E+06  -4.788095E+05
2      3.853866E+06  -3.824006E+06   4.788095E+05
3      4.396770E+06  -4.396770E+06   0.000000E+00
4      5.423444E+06  -5.423444E+06   0.000000E+00
5      6.435558E+06  -5.867538E+06  -2.643560E+06
6      6.435558E+06  -5.867538E+06   2.643560E+06
7      6.554762E+06  -3.126871E+06  -5.760866E+06
8      6.554762E+06  -3.126871E+06   5.760866E+06
9      1.343013E+07  -1.343013E+07   0.000000E+00
10     1.531050E+07  -1.531050E+07   0.000000E+00
11     2.217833E+07  -2.217833E+07   0.000000E+00
12     6.564085E+07  -6.564085E+07   0.000000E+00
13     1.228260E+08  -1.228260E+08   0.000000E+00
14     3.857487E+06   3.857487E+06   0.000000E+00
15     2.908742E+09   2.908742E+09   0.000000E+00
```

At this point it is possible to simplify the results, since in practice many poles and zeros will be output and some of them will be almost superimposed giving a negligible resultant effect. Moreover, sometimes the circuit will work in a well determined frequency range over which we will focus our attention, while also all the parasitic and less meaningful roots will be detected. That is why the following reduction mechanisms are available.

Frequency Limit

The program asks the user if he wants to limit the analysis up to a certain frequency. Here is such a dialog:

```
Do you want to set an upper frequency limit for the selected poles and
zeros? [yes/no]:
y
So give the highest frequency to be considered:
5e7
Roots up to 0.5000E+08 Hz will be examined.
```

If the roots of your circuit are spanned up to a frequency quite a lot higher than the upper frequency of your AC analysis, this limit can be very useful in simplifying textual output and FNS models without meaningful loss of accuracy.

Pole-Zero Cancellation by Threshold

Very close poles and zeros can be deleted as their influence on the circuit behavior compensates. Big circuits with a lot of roots often have closely spaced poles and zeros which can be deleted to show the most meaningful remaining roots. A prompt reminds the user about the threshold mechanism, showing the condition to delete a pole **P** and a zero **Z**. The condition is that:

```
abs(RE[P]-RE[Z]) < abs(RE[P]) ◊ TH + TH
```

and the same for the imaginary part.

Therefore, if a threshold of say, 0.1 is given, there are two possibilities:

- For very low frequency roots, say fractions of 1 Hz, `TH` has the meaning of maximum difference between the real parts of the pole and the zero, and the same for the imaginary part.
- In most cases, there are roots at quite high frequencies so the first term on the right side of the relation dominates and `TH` takes the meaning of maximum ratio between the distance amongst the roots and their value, so in this case a 10% tolerance.

At this point, a table of remaining poles and zeros (after the two elimination steps) is given.

Hand Selection

After these steps, the user can still express a further choice between the resulting roots. A prompt asks:

```
Do you agree with the Selected POLES [Yes/No]?
```

If the selected poles and zeros are Ok, then press `Y` and the analysis goes on. If only a certain number of poles and zeros are required, press `N` to reveal the following prompt:

```
Poles selection by their index in INCREASING order
```

Enter a negative index to end the selection.

Select Index

The user is asked to select which poles are to be included in the final transfer function. The selection is made via the index to the values—the index being the first number which appears in the tables above. Once the values to use have been chosen, the selection is terminated by entering a negative number. The same dialog then starts for zeros.

Note

Make sure the selections are made in increasing integer order including the negative terminator otherwise required values may be lost.

The remaining set of poles and zeros are printed, followed by another table showing the difference between the actual and the now approximated model, swept in frequency and giving both the magnitude (dB) and the phase (degree) differences. This is shown below:

```
Hz              dB’s Error    Degrees Error
1.000000E+01  6.525140E-04  1.020462E-02
1.000000E+02  6.201058E-04  1.096881E-02
1.000000E+03  5.805557E-04  1.102087E-02
1.000000E+04  5.806112E-04  1.107192E-02
1.000000E+05  5.791327E-04  3.599889E+02
1.000000E+06  5.815470E-04  1.444244E-02
1.000000E+07  1.179731E-03  4.588650E-02
1.000000E+08  5.953287E-02  3.609858E-01
1.000000E+09  4.490041E+00  2.134625E+00
```

FNS Model

The reduced circuit is now modeled as an S-domain transfer function (say an FNS device in Eldo syntax) which can replace the original circuit with the introduced approximations for AC and small signal analysis—this can be very useful for developing complex designs, allowing the replacement of even a complex block with its equivalent model and great simplification of simulation of the higher level system. The output format is shown below:

```
FNS1  IN OUT
+
+      498057.650621
+    0.0271151244556  !   * s^(-1)
+  4.88425501019e-10  !   * s^(-2)
+  2.65330374224e-18  !   * s^(-3)
+ -3.22583377778e-27  !   * s^(-4)
+      8.84119411e-37  !   * s^(-5)
+ ,
+                  1
+    0.0315272851854  !   * s^(-1)
+  1.08957721435e-09  !   * s^(-2)
+  1.85561098689e-17  !   * s^(-3)
+  2.03791858143e-25  !   * s^(-4)
+  1.03556420914e-33  !   * s^(-5)
```

If too many poles and zeros are left after the reduction steps, FNS evaluation could fail due to too high coefficients (a message is printed in such a case). However, the efficiency of the reduction algorithm normally allows a reduction of at most 6-7 poles which can give a very good approximation of the original circuit.

The PZ program produces a *<circuit>.cpz* output file to be visualized with EZwave which shows the Bode diagrams of gain and phase for the original and the reduced circuit. Also, a file *<circuit>.mpz* is written, showing a scatter map of the circuits roots in the complex plane. And finally, a subcircuit containing FNS is stored in file *<circuit>.pzck*, which can be included by an upper level netlist.

After all these steps, the user is asked whether they wish to repeat the analysis, typically to alter some parameters such as threshold, frequency limit, and so on, thereby modifying the accuracy/simplicity ratio of the whole analysis.

When “no” is the answer, the program is exited leaving an ASCII output file *<circuit>.pzr* containing a trace of all the program outputs and the dialog.

Pole-Zero Numerical Issues

There are possible numerical issues with poles and zeros. This section explains why 500 variables is a reasonable maximal system size, and suggests a workaround using a two-step S-parameter block approach.

The **`.PZ`** command generates G and C matrices. The pole-zero post-processor computes poles and zeros, and generates approximated FNS to be used. It uses a QZ algorithm to solve the generalized Eigenvalue system:

$$GX = \lambda CX$$

The CPU cost of the QZ algorithm is approximately n^3, where n is the size of the G and C matrices. When $n > 500$, the CPU time is large, and numerical issues may arise. This is a constraint of the QZ algorithm.

Workaround: Use a two-step S-parameter block approach:

1. Extract the S-parameters of the circuit (using **`.FFILE`** and **`.AC`** commands)

   This generates a *.s2p* file containing the S-parameters.

2. Use this S-parameter file in either of these ways:

   - Use this S-parameter file (*.s2p*) as a macromodel of the circuit (using an FBLOCK model)

     This is more accurate than the FNS pole-zero approach.

   - Alternatively, specify the **`.PZ`** command with the S-parameter file (*.s2p*) (using an FBLOCK model)

     The pole-zero post-processor can then be used to compute the poles and zeros as before, avoiding any numerical issues.

Chapter 18
Optimizer in Eldo

Introduction

Overview

The Eldo optimizer is a general-purpose electrical circuit optimization program. The optimizer will calculate the value of parameters (the optimization variables) in the circuit such that the behavior or the characteristics of the circuit conform as close as possible to the specifications. The optimizer can achieve a simultaneous improvement in AC, DC, Transient domain, Steady-State and Modulated Steady-State analyses.

The designer specifies the design objectives and the optimizer will adjust the component parameters of the circuit (such as resistor or capacitor values, the β value of a transistor, widths and lengths of a MOSFET) in order to meet a specified electrical performance. Optimization can be applied to:

- Circuit parameters
- Model parameters
- Element parameters
- Device lengths, widths, areas, and peripheries.

The parameter values must conform to the manufacturing limits, process limits, or discrete device values. To achieve this, restrictions can be specified on the design parameters. Various constraints (or inter-relations) can be specified, for example, circuit parameters must be non-negative, or must not violate upper boundaries. In addition, more complicated constraints can be specified, for example, how components physically interact with each other to produce non-linear relations.

The process of identifying the *objective*, *variables*, and *constraints* for a specific problem is known as *modeling*. The construction of an appropriate model is the first step (sometimes the most important) in the optimization process. If the model is too simplistic, it will not generate useful insights into the practical problems. If too complex, it may become difficult to solve. Thus, the designer's task is to discover what is the most appropriate model for their requirements for more information please refer to "Modeling Capabilities in Eldo and Optimization" on page 1165.

The term Eldo optimizer refers to the complete set of optimization tools available through the **.OPTIMIZE** command. In some circumstances it is necessary to be more precise about the underlying method or algorithm, therefore the nomenclature in Table 18-1 will be used.

Table 18-1. Nomenclature

Name	Algorithm/Method	Command
Eldo optimizer/Passfail	Pass or fail technique	**.OPTIMIZE METHOD=PASSFAIL**
Eldo optimizer/Bisection	Dichotomy (or bisection) algorithm	**.OPTIMIZE METHOD=BISECTION**
Eldo optimizer/Secant	Hybrid bisection/Secant	**.OPTIMIZE METHOD=SECANT**
Eldo optimizer/SQP	SQP method	**.OPTIMIZE**
Eldo optimizer/Search	Bisection and inverse quadratic interpolation	**.OPTIMIZE METHOD=SEARCH**

This documentation is mainly dedicated to the Eldo optimizer/SQP. The other group of methods (Bisection, Secant and Passfail) represent a specific class of algorithms pertaining to the category of Derivative Free Optimization (DFO) algorithms. They allow optimization in the restricted case of *one dimensional* problems. These methods will be documented as special cases of the default algorithm.

Problem Statement

In order to use the Eldo optimizer, the designer must provide the following information:

- The *nominal circuit*, identical to the circuit provided for simulation in the netlist format. The user must have a working netlist.
- The *design variables*, the designer must specify those parameters that may be optimized by the optimizer in its search for an optimal solution. The designer's selection of variables is accomplished by making minor modifications to the working netlist, using the **.PARAMOPT** command. Please refer to ".PARAMOPT" on page 1142.
- The *design objectives* may be any quantity represented by a real value, a combination of quantities that are generated by multiple sweep commands, or multiple increment step on circuit parameters. For example, a profile describing the frequency response desired for a filter circuit. Eldo provides a number of methods for specifying a waveform to match, or by simply defining point(s) that correspond to the desired behavior. These design objectives are specified by adding data and simple arguments to the **.OBJECTIVE** command (or the **.EXTRACT** and **.MEAS** commands). This important point is explained in "Modeling Capabilities in Eldo and Optimization" on page 1165.

The information that the user has to provide is referred to as the *problem statement*. The designer provides a model for the optimization problem, this may be the minimization or maximization of functions (extracted measures) subject to the *constraints* on its variable.

For example, the **.PARAMOPT** command is used to specify the length `l` and the width `w` of a MOS transistor:

```
.PARAMOPT
+ L=(10U, 2U, 100U)
+ W=(60U, 2U, 200U)
```

These commands can be expressed mathematically as the inequalities:

$$2 \times 10^{-6} \leq l \leq 10^{-4}$$

and

$$2 \times 10^{-6} \leq w \leq 2 \times 10^{-4}$$

The first value given in the command **.PARAMOPT L**=(10U,2U,100U) is used to initialize the optimization algorithm.

On the other hand, the objective function will be specified by adding a **.OBJECTIVE** command. For example, the optimization of the phase margin for an operational amplifier in closed-loop configuration:

```
.OBJECTIVE AC
+ LABEL=phasemarg ( XYCOND(VP(out),VDB(out) < 0) + 180 )
+ LBOUND=50.0
```

Once the design parameters and the design objective have been defined, the user must add the **.OPTIMIZE** command to the netlist file, and execute Eldo.

Invoking the Eldo Optimizer

The Eldo optimizer can be invoked from the command line in the same way as Eldo:

```
eldo <circuit_name> <other_arguments>
```

Exploiting the Results of Optimization

After the optimization algorithm has been applied, the designer must be able to recognize whether it has succeeded in its task of finding a solution. This can be accomplished by analyzing the results of the optimization.

- There are the final diagnostics for checking that the current set of variables is a solution of the problem. If the optimal conditions are not satisfied, they may give useful information on how the current estimate of the solution can be improved. Users may think of this as a *global score* of the optimization run.
- There are also the values of the extracted measures printed during the optimization our at the end of the process. This represents the *post-optimization* analysis.

The designer will find this kind of information within the output results of Eldo (*.chi* file) and the Eldo optimizer (*.otm* file). The script `viewotm` can be invoked to extract the desired results. This script can be invoked from the command line as follows:

```
viewotm -f <circuit_name>.otm <other_arguments>
```

Please refer to "Exploiting Output Results" on page 1190 for more information.

From the optimizer's point of view, it is important to understand that the only way to check optimality (or success) is to compute some measure of optimality and feasibility at the final solution. There is no other criterion (the optimizer is not an expert in circuit design). If this optimality measure is less than some specified tolerance, the current point is considered as optimal. The fundamental notions of optimality and feasibility are considered in a semi-rigorous way along this documentation, for more information please refer to "Role of tolerances in Eldo optimizer/SQP" on page 1186. These ideas are presented as concise as possible, but it is not mandatory if the user wishes to avoid technical details.

Optimization Commands

The following are commands that can be used for optimization:

- .EXTRACT and .MEAS
- .OBJECTIVE
- .OPTIMIZE
- .PARAMOPT

In the descriptions, we will use the tokens `IVALUE` and `RVALUE` to denote integer and real (or floating) numbers.

.EXTRACT and .MEAS

In the context of optimization, this command is an extension of the **.EXTRACT** command. The additional arguments (**GOAL**, **EQUAL**, **LBOUND**, **UBOUND**, and **WEIGHT**) have no effect when the **.OPTIMIZE** is not specified in the netlist. The documentation of the **.EXTRACT** command defines two categories of design objectives:

- those represented by a single number (scalar version)
- the objectives represented by a vector of measures.

These two categories are described in the following sections with some known limitations.

Command Syntax for Scalar Objectives

The general specification can be written as:

```
.EXTRACT
+ [EXTRACT_INFO] [LABEL=NAME] [FILE=FNAME]
+ [VECT] [CATVECT] $MACRO|FUNCTION
+ [OBJECTIVE_INFO]
```

Parameters

- **[EXTRACT_INFO] [LABEL=NAME] [FILE=FILE_NAME] [VECT] [CATVECT]**

  Please refer to ".EXTRACT" on page 637 for more information.

- **$MACRO | FUNCTION**

  Instantiation of a macro (preceded by the '$' character), or an expression that uses the Extraction Language of Eldo.

- OBJECTIVE_INFO:=

  ```
  GOAL=MINIMIZE|MAXIMIZE [WEIGHT=RVALUE]
  | GOAL=RVALUE [WEIGHT=RVALUE]
  | EQUAL=RVALUE
  | {LBOUND=RVALUE | UBOUND=RVALUE}
  | LBOUND=RVALUE UBOUND=RVALUE
  ```

  The OBJECTIVE_INFO argument describes the category of the design objective when optimization is required. When this argument defines a **MINIMIZE** or **MAXIMIZE** objective, the specified expression will be minimized or maximized. The argument **GOAL**=RVALUE defines a soft constraint on the measure, while the **LBOUND**, **UBOUND** and **EQUAL** define hard constraints. The specification of the arguments **GOAL**, **EQUAL**, **LBOUND**, and/or **UBOUND** is mandatory for optimization. The default value of **GOAL** when none is defined is 0; this does not apply to the passfail method. The **WEIGHT** argument is a positive value attached to the design objective. The **WEIGHT** can be thought of as quantifying the user's desire to make the objective important or not.

Command Syntax for Vector Objectives

The specification of vector objectives is performed with the **.EXTRACT FITTING** command.

```
.EXTRACT
+ [EXTRACT_INFO] [LABEL=NAME] [FILE=FILE_NAME]
+ [VECT] [CATVECT]
+ FITTING
+ TARGET_NAME $MACRO|FUNCTION
+ [SCALE_INFO] [WEIGHT=RVALUE]
+ [INTERP_INFO | SELECT_INFO]
```

Parameters

- **[EXTRACT_INFO] [LABEL=NAME] [FILE=FILE_NAME] [VECT] [CATVECT]**

  Please refer to ".EXTRACT" on page 637 for more information.

- **TARGET_NAME**

  Name of the discretized curve to fit. It can be the name of a previously defined wave or the reference to a variable in a **.DATA** statement (for more information run the example *fourband* in the directory: *$MGC_AMS_HOME/examples/optimizer*).

- **$MACRO|FUNCTION**

  Instantiation of a macro (preceded by the '$' character), or an expression that uses the Extraction Language of Eldo.

- **SCALE_INFO**:= [**LINEAR**|**LOG**|**LOG10**]

  The **LINEAR**, **LOG**, and **LOG10** select the scale of the x-axis for the extracted measures.

- **INTERP_INFO**:= **INCR**=RVALUE

  The **INTERP_INFO** argument is meaningful only when goal values are specified through **.DATA** statements. The **INCR** value will be the step in the analysis sweep variable to be used for generating goal values by a piece wise linear interpolation of data provided by an associated **.DATA** command.

- **SELECT_INFO**:= **DEC**=RVALUE

  The **DEC** argument is the number of points per decade in the analysis sweep variable to be used for generating goal values using a logarithmic interpolation of the provided data points. The corresponding simulated values are obtained by interpolation of the available simulated values.

Notes

- The user can specify a multi-dimensional specification of **GOAL** objectives, this is equivalent to specifying the **.EXTRACT FITTING** command.

- Coherency has been enforced between analysis commands and objective specifications. The following example is representative. Some **DC** operating point analyses are performed for every value of VDS from 0 to 40 with a step increment of 1, and every value of VGS between 4 and 16 with a step increment of 2. Three design objectives are defined for the variable IDS through the **.DATA** labeled MOSFIT, corresponding to the values of VGS equal to 4 and the values of VDS equal to 1, 2, and 3 respectively. The current at node VDRAIN is optimized to fit the values of IDS. The values of VDS and

VGS for the measurements correspond to the simulation points on a short range as specified in the **.DC** command.

```
.DC VDS 0 40 1 VGS 4 16 2

.EXTRACT DC LABEL=FIT_IDS
+ FITTING
+ MOSFIT(IDS) I(VDRAIN)

.DATA MOSFIT
+ VDS VGS IDS
+ 1.000E+00 4.000E+00 1.574E+00
+ 2.000E+00 4.000E+00 1.806E+00
+ 3.000E+00 4.000E+00 1.826E+00
.ENDDATA
```

Such situation is now detected and forbidden. In this case, the analysis command should be driven by the values of VDS and VGS specified in the **.DATA**. The use of **.DATA** constructs is then restricted.

Consider the next example. A small signal analysis is performed for frequencies in the range [100 kHz, 1 GHz] with 2 points per decade. The voltage magnitude at node 16 is optimized to fit the values of the data labelled V16 in the statement **.DATA** VM16.

```
.AC DEC 2 100k 1g

.EXTRACT AC LABEL=FIT_VM16
+ FITTING
+ VFIT(V16) VM(16)

.DATA VFIT
+ FREQ V16
+ 100K 200
+ 30Meg 200
+ 1G 1m
.ENDDATA
```

There are three measurement points defined at 100 kHz, 30 MHz and 1 GHz respectively. In the former version of Eldo (v6.6), the values of VM(16) at these frequencies were obtained by interpolation of the values simulated according to the **.AC** analysis command. Interpolation for measures is now forbidden with optimization, since this technique introduces additional errors that cannot be bounded, and gives non-smooth measurements. When optimization is required the simulation points must be included in the set of simulation points.

.OBJECTIVE

The **.OBJECTIVE** command is based on the **EXTRACT** construct, and specifically dedicated to optimization. Some restrictions are imposed and the semantic is different in some situations. The specification of design objectives has been simplified. The following is a list of reasons of using the **.OBJECTIVE** command:

- Some implicit rules in the **EXTRACT** construct are not well fitted for optimization. For example, when several **ALTER** blocks are used with optimization, the **.EXTRACT** commands are simply added to each **ALTER** blocks. The notion of *scope* related to a **.EXTRACT** command was not clearly defined. By default, the **.OBJECTIVE** commands have a local scope.
- It is easier to separate the optimization block, the circuit statements, and the plot statements.
- In the *.chi* file, it is difficult to print the values of objectives as they are effectively processed in the optimizer.
- The dump of the results into a file (using the **FILE** argument) is not of great use. Moreover, vector optimization makes the results difficult to read.
- The actual version of this command is limited to the basic analyses such as DC, TRAN, and AC. This list will be completed in the subsequent releases of the optimizer.

Note

For backward compatibility, the equivalent of the **.OBJECTIVE** command can be specified with the **.EXTRACT** command.

Command Syntax

The general specification can be written as:

```
.OBJECTIVE
+ EXTRACT_INFO [LABEL=NAME]
+ {$MACRO|FUNCTION}
+ OBJECTIVE_INFO
+ [SCALING_INFO]
+ [PRINT_INFO]
```

Parameters

- [EXTRACT_INFO] [**LABEL**=NAME]

  The argument EXTRACT_INFO must be replaced by one element of the following list of analyses: **DC**, **DCTRAN**, **TRAN**, **DCAC**, or **AC**. An analysis can be omitted, in this case the argument $MACRO|FUNCTION must represent a valid expression that refers to other extracted measures.

- **$MACRO** | **FUNCTION**

  Instantiation of a macro (preceded by a '$' character), or expression that uses the Extraction Language of Eldo.

- OBJECTIVE_INFO:=
  GOAL=**MINIMIZE**|**MAXIMIZE** [**WEIGHT**=RVALUE]
  | **GOAL**=RVALUE [**WEIGHT**=RVALUE]
  | **EQUAL**=RVALUE
  | {**LBOUND**=RVALUE | **UBOUND**=RVALUE}
  | **LBOUND**=RVALUE **UBOUND**=RVALUE

  The **OBJECTIVE_INFO** argument describes the category of the design objective when optimization is required. When this argument defines a **MINIMIZE** or **MAXIMIZE** objective, the specified expression will be minimized or maximized. The argument **GOAL**=RVALUE defines a soft constraint on the measure, while the **LBOUND**, **UBOUND** and **EQUAL** define hard constraints (see "Notes" on page 1136 for the definition of *hard* and *soft constraints*) The specification of the arguments **GOAL**, **EQUAL**, **LBOUND**, and/or **UBOUND** is mandatory for optimization. The **WEIGHT** argument is a positive value attached to the design objective. The **WEIGHT** can be thought of as quantifying the user's desire to make the objective important or not.

- **SCALING_INFO**:= **TYPVAL**=RVALUE

  This argument allows the user to specify the typical value or the scale factor associated to the objective. Please refer to "Scaling of variables and objectives" on page 1177 for more information.

- MONITOR_INFO:= **MONIT_VAL** [=**QUALIFIER** {, QUALIFIER}]]

  where QUALIFIER=**MAX**|**MIN**|**MEAN**|**ERRMAX**|**ERRMIN**|**ERRMEAN**

  MAX The maximum value of the multi-point objectives. Optional.

  MIN The minimum value of the multi-point objectives. Optional.

  MEAN The mean value of the multi-point objectives. Optional.

  ERRMAX The maximum value of the error functions for the multi-point objectives. Optional.

  ERRMIN The minimum value of the error functions for the multi-point objectives. Optional.

  ERRMEAN The mean value of the error functions for the multi-point objectives. Optional.

  The **MONIT_VAL** argument allows the user to specify how the current values of a design objective are printed during optimization. This option affects only the *.otm* file (the Eldo optimizer output) but not the *.chi* file or the standard output. It is possible to have multiple options, they should be separated by commas, for example: **MONIT_VAL**=**MAX**,**ERRMAX**... Default value is **MEAN**. See "How to monitor the design objectives (MONIT_VAL options)" on page 1195 for the details.

- **PRINT_INFO**:= **PRN_VAL**={**NO**|**YES**}

  The **PRN_VAL** argument allows the user to specify whether the values of a specific objective must be printed. By default **PRN_VAL**=**YES**, all the components of a design objective are printed.

Notes

- The additional data such as the goal values, the lower/upper bounds, and the weight numbers are sent to the optimizer when initializing the optimization process. They have constant values during the whole process.
- Excepting the design objectives specified with **GOAL**=MINIMIZE|MAXIMIZE, which play a specific role in the problem statement, the goal values and the bounds have *multi-point* extension. This feature is related to data driven analysis. For an example please refer to “Effect of multiple sweeps and step increments” on page 1182.
- Note that some of the arguments in **OBJECTIVE_INFO** are mutually exclusive. For instance, an objective cannot be specified at the same time as an equality and as an inequality. The user may think of as an extracted measure having a unique descriptor that gives its role within the problem.

  These rules are restrictive. From a mathematical viewpoint, it makes sense to minimize and bound a design objective, or to specify a goal value and a range of values for an objective. This must be done on separate commands.
- In general not all equality and inequality constraints are perceived by designers in the same way. It leads to classify constraints as either *hard* or *soft*.

  The term *hard constraints* means refers to the constraints the designer considers as most essential, i.e. they have to be satisfied. *The designer does not want them to take part in any subsequent design trade-off*. For example, any constraint required for physical reliability must be treated as a hard constraint.

  In contrast, *soft constraints* are those that *the designer is interested in trading off against one another* and against the performance objectives during intermediate iterations of an optimization run. Note that **MINIMIZE** and **MAXIMIZE** objectives are naturally candidates for trade-off.
- The Eldo optimizer/SQP method is the only algorithm supporting the complete set of arguments in **OBJECTIVE_INFO**. For example, the statement:

  ```
  .OBJECTIVE {...} GOAL=MINIMIZE|MAXIMIZE
  ```

  have no sense when used with the Dichotomy and Secant methods. These methods are only dedicated to solve equations of the form F(x)=0. An error message will be raised in these cases.

These restrictions are described in Table 18-2:

Table 18-2. .OBJECTIVE Restrictions

Statement	Notation	Goal Value	Lower Value	Upper Value	Weight Value	Supported Methods
.OBJECTIVE {...} + **GOAL**=**MINIMIZE**	$f_m(x)$	N/A	N/A	N/A	μ_m	SQP, Search
.OBJECTIVE {...} + **GOAL**=**MAXIMIZE**	$-f_m(x)$	N/A	N/A	N/A	μ_m	SQP, Search
.OBJECTIVE {...} + **GOAL**=r	$f_r(x)$	r	N/A	N/A	μ_r	SQP, Search, Secant, Dichotomy
.OBJECTIVE {...} + **EQUAL**=e	$f_e(x)$	e	N/A	N/A	Not used	SQP, Search, Secant, Dichotomy
.OBJECTIVE {...} + **LBOUND**=l + **UBOUND**=u	$f_i(x)$	N/A	l	u	Not used	SQP

.OPTIMIZE

This section is concerned with the optimization command acting on the behavior of the Eldo optimizer algorithms. In some specific cases, designers can perform optimization using the DICHOTOMY, the SECANT and PASSFAIL methods. Eldo can run the optimizer without using the default algorithm Eldo optimizer/SQP. This can be applied when there is only one goal (specified through a **GOAL**=RVALUE statement) and one parameter to optimize. Note that the **.OPTIMIZE** command is compatible with the multiple run features (see the command description for ".MPRUN" on page 729).

Command Syntax

The optimization specification acting on all the analyses specified in the circuit netlist is done using the following command:

```
.OPTIMIZE
+ [METHOD=DICHOTOMY|SECANT|PASSFAIL|SEARCH]
+ [QUALIFIER=VALUE {, QUALIFIER=VALUE}]
+ [PARAM=LIST_OF_VARIABLES|*]
+ [RESULTS=LIST_OF_MEASURES|*]
+ [OUTER=LIST_OF_PARAMETERS]
```

Parameters

- **QUALIFIER**

  The name of the corresponding configuration argument (see Eldo Optimizer/SQP Arguments).

- **PARAM**=LIST_OF_VARIABLES

  List of comma-separated variables to be tuned, specified with the **.PARAMOPT** command.

- **RESULTS**=LIST_OF_MEASURES

  List of comma-separated measures to be optimized, specified with **.EXTRACT** and/or **.MEAS** commands.

- **OUTER**=LIST_OF_PARAMETERS

  List of comma-separated design parameters specified with the **.PARAM** command and used in a **.STEP PARAM** command. For more information, please refer to "Outer and inner design parameters" on page 1170.

Note

If the specification of design variables with **PARAM**= is omitted or if the character * is specified in place of an explicit list, all the variables specified by a **.PARAMOPT** command are optimized. If the specification of objectives with **RESULTS**= is omitted or if * is specified in place of an explicit list, then all the design objectives specified by a **.OBJECTIVE (or a .EXTRACT**/**.MEAS** command) are optimized.

Eldo Optimizer/SQP Arguments

The performance of the solver (the SQP algorithm) is controlled by a number of parameters. Each option has a default value that should be appropriate for most problems. The defaults are given below. For specific situations it is possible to specify non-standard values for some or all of the parameters. If experimentation is necessary, it is recommend that the user only changes one option at a time.

- **MAX_ITER**=IVALUE

  Maximum number of iterations permitted for optimization. Default 1000.

- **MAX_SIMUL**=IVALUE

  Maximum number of circuit simulations allowed. Default 99999.

- **TOL_OPT**=RVALUE

  Tolerance on optimality conditions. This parameter specifies the accuracy to which the user wishes the final iterate to approximate a solution of the problem. **TOL_OPT** can be considered as indicating the level of accuracy (i.e. the number of significant figures) desired in the design functions at the solution. Specifying only the value **TOL_OPT** has the same effect as setting separately both the tolerances **TOL_GRAD**=**TOL_OPT** and **TOL_FEAS**=$10^{-2} \times$ **TOL_OPT**. Default value is 10^{-4}.

- **FD_FWRD_INT**=RVALUE

  Relative design-variable change for the computation of the perturbation used for gradient evaluations. The perturbations are taken as **FD_FWRD_INT** times the maximum of the absolute design parameter. Default value = 10^{-6}.

- **TOL_FEAS**=RVALUE

  Tolerance on feasibility conditions. An iterate is said to satisfy the feasibility conditions if $\mathrm{FEAS}(x) \le \mathrm{TOL}_{\mathrm{FEAS}} \times \mathrm{SIZE}\ (x)$, where SIZE(x) represents a scaling quantity taking into account the norm of the solution vector. Default value is 10^{-6}.

- **TOL_GRAD**=RVALUE

  Tolerance on the measure of *criticality* of the current iterate. An iterate is said to be *critical* if $\mathrm{OPTIM}(x) \le \mathrm{TOL}_{\mathrm{GRAD}} \times \mathrm{SIZE}\ (\lambda)$ where SIZE(λ) is a scaling factor representing the sensibility of the current solution with respect to changes of problem data. Default value is 10^{-4}.

- **PRN_MAJOR=YES|NO**

  Controls the output of the optimization algorithm. Several lines of informations are written to the *.otm* file during the optimization process. This output is a structured file that can be examined with the `viewotm` script. When set to **NO**, the information will not be written to the file. Default is **YES**. For more information please refer to "Exploiting Output Results" on page 1190.

- **PRN_MINOR**=**YES|NO**

  Controls the output of the optimization algorithm at a low level of description. A large amount of information is written to the *.qtm* file at each major iteration. This file does not reveal proprietary informations about the user's problem. It describes only the optimization run, and is dedicated to support. The structure of this file may change significantly between releases, it is strongly recommended to not base scripts or wrappers on this file. Default is **NO**.

Eldo Optimizer/Search Arguments

When **METHOD**=**SEARCH** the *Search* method is launched. This method can be controlled with the following arguments:

- **MAX_ITER**=IVALUE

  Maximum number of iterations allowed for the algorithms.

- **TOL_RELPAR**=RVALUE

  Convergence criterion on the relative variation of the optimization variables. Note that this argument exists for the Secant/Dichotomy/Passfail methods. The stopping test is implemented as follows. We measure the relative change in x by the quotient:

$$relx = \frac{|x_{new} - x_{cur}|}{max\{|x_{new}|, typx\}}$$

  where x_{new} is the new value of the variable, x_{cur} is the current value of the variable, and typx is the nominal or typical value of x (which is computed dynamically). The test is satisfied if one has:

$$relx \leq TOL_{RELPAR}$$

  For the selection of **TOL_RELPAR** one could adopt the basic rule: if p significant digits of the optimal solution are desired, **TOL_RELPAR** should be set to 10^{-p}. The default value is 10^{-3}.

Eldo Optimizer/Dichotomy/Secant/PassFail Arguments

The optimization specification acting on one-dimensional problems is achieved by specifying the argument **METHOD**=**DICHOTOMY|SECANT|PASSFAIL**. The following is a list of arguments that are supported with these algorithms:

- **MAX_ITER**=IVALUE

  The maximum number of iterations allowed for the algorithms.

- **TOL_RELPAR**=RVALUE

  Convergence criterion on the relative variation of the optimization variables. The optimization is stopped when the relative change at the current iteration is approximately less than **TOL_RELPAR**. This test is performed with:

$$relx = \frac{|x_{new} - x_{cur}|}{|x_u - x_l|}$$

and the algorithm is stopped when:

$$relx \le TOL_{RELPAR}$$

Default value is 10^{-3}.

- **TOL_RELTARG**=RVALUE

  Convergence criterion on the relative change of the design objective. This test is performed with:

  $$|f(x_{new}) - f(x_{cur})| \le TOL_{RELTARG} \cdot |f_{GOAL}|$$

  where f_{GOAL} is the goal value. Default value is 10^{-3}.

Note

The previous convergence tests have some drawbacks. For example, the last one is never satisfied when the goal value is *exactly* zero (except when $f(x_{new}) = f(x_{cur})$). A future release will correct this issue.

- **LIMIT**=PASS|FAIL

  Only used with the Passfail method. If **LIMIT** is set to PASS (default), the last iteration is always the one for which the measurement is correct. If **LIMIT** is set to FAIL, the last iteration will be the one for which the measurement fails.

.PARAMOPT

The specification of the optimization variables is realized with an extension of the **.PARAM** command. These variables will be denoted by a vector of real values of dimension N:

$$x = (x^{(1)}, x^{(2)}, ..., x^{(N)})$$

The vector x is also associated with vectors of the lower and upper bounds denoted by x_1, x_u. An additional vector δ will be associate to the discretization (or resolution) of the variables x.

Command Syntax

The design variables must be specified through the **.PARAMOPT** command:

```
.PARAMOPT VARIABLE_NAME=(
+ [INIT_VALUE,]
+ {LOWER_BOUND | LOWER_PERCENT% },
+ {UPPER_BOUND | UPPER_PERCENT% }
+ [, INCREMENT])
```

Parameters

- VARIABLE_NAME

  Name of the design variable(s). This parameter can be one of the following:

  ```
  PARAMETER_NAME |
  P(PARAMETER_NAME) |
  P(SUBCKT_NAME,SUBCKT_PARAMETER) |
  E(DEVICE,PARAMETER) |
  M(MODEL_NAME,MODEL_PARAMETER) |
  EM(DEVICE_NAME,MODEL_PARAMETER)
  ```

  The character strings PARAMETER_NAME, SUBCKT_NAME, and MODEL_NAME may contain wildcard characters * and ?. These features allow a *group* of parameters to be manipulated, rather than a *unique* parameter i.e. the **.PARAMOPT** command has two distinct behaviors; it can define the vector of design parameters x, using a *component-wise* specification, or a *generic* specification. These features are described below in Generic Specification of Design Parameters.

- INITIAL_VALUE

  Initial value of the design variable. It is optional in some situations, see Optional Initial Value.

- LOWER_BOUND, UPPER_BOUND

  Lower and upper bounds specified for the design variable. Unbounded (or free variables) must be specified using the star character *, for example:
  .PARAMOPT VAR1=(1.0, 0.0, *)
  specifies a non- negative $0 \leq var1 \leq \infty$ parameter VAR1 with the initial value set to 1.0. Free variables $-\infty \leq x \leq \infty$ are specified using the triplet:
  .PARAMOPT VARIABLE_NAME=(INIT_VALUE, *, *)

- `LOWER_PERCENT%`, `UPPER_PERCENT%`

  Percentages of the initial value. For example, the command:
  .PARAMOPT VAR1=(5.0, 10%, 35%), specifies that the effective lower and upper bounds are $x_l^{(1)} = 5.0 \times (1 - 0.1)$ and $x_u^{(1)} = 5.0 \times (1 + 0.35)$.

- `INCREMENT`

  Specifies a discretization for the final value of the design parameter, see Discretization of Design Variables for more information. Optional.

Caution

The **.PARAMOPT** command supports fixed variables: $x_l^{(i)} = x^{(i)} = x_u^{(i)}$. For instance, it is possible to fix the number of fingers N1 when optimizing a NMOS device by specifying: **.PARAMOPT** N1= (30, 30, 30). The simulation time is proportional to the number of variables. For efficiency, this feature should be avoided.

For example, to optimize the length **L** and width **W** of a MOS transistor, the parameter specification can be:

```
.PARAMOPT
+ L = (10U, 2U, 100U, 0.01U)
+ W = (60U, 2U, 200U, 0.01U)
```

Here, the optimization is initiated with a length of 10µm and a width of 60µm The length is allowed to change between 2µm and 100µm by steps that are multiples of 0.01µm. Similarly the width can to take values between 2µm and 200µm, that differ from the initial value (60µm) by a multiple of 0.01µm.

Specification of Design Parameters

Suppose that an optimization has to be performed on the following circuit:

```
.MODEL NMOD1.1 NMOS LEVEL=53
+ LMIN=0.1U LMAX=0.2U
+ WMIN=1U WMAX=100U
+ VTH0=0.2

.MODEL NMOD1.2 NMOS LEVEL=53
+ LMIN=0.2U LMAX=0.3U
+ WMIN=1U WMAX=100U
+ VTH0=0.3

.MODEL NMOD1.3 NMOS LEVEL=53
+ LMIN=0.3U LMAX=1U
+ WMIN=1U WMAX=100U
+ VTH0=0.4

VD D 0 DC 1.5
VS S 0 DC 0
VB B 0 DC 0
VG G 0 DC 3

RD1 D D1 1
```

```
RD2 D D2 1
M1 D2 G S B NMOD1 W=10U L=0.25U
M2 D1 G S B NMOD1 W=10U L=0.35U
```

Adding the following commands will *only* change the length of `M2`:

```
.OPTIMIZE
.PARAMOPT E(M2,L)=(0.35U, 0.1U, 1U)
```

The length of `M1` and `M2` can be changed with:

```
.OPTIMIZE
.PARAMOPT E(M*,L)=(*, 0.1U, 1U)
```

The initial values specified on the instances are kept unchanged. The following command will change lengths of `M1` and `M2`, the initial values are both set to 0.35μ.

```
.OPTIMIZE
.PARAMOPT E(M*,L)=(0.35U, 0.1U, 1U)
```

The following sequence of examples will affect the parameter `VTH0`.

```
.OPTIMIZE
.PARAMOPT EM(M2,VTH0)=(0.3, 0.1, 1)
```

Changes the parameter `VTH0` of the model used by `M2`.

Note

For this kind of parameter, the initial value is ignored.

Using wildcards, the parameter `VTH0` can be changed for both `M1` and `M2`, this can be achieved with:

```
.OPTIMIZE
.PARAMOPT EM(M*,VTH0)=(*, 0.1, 1)
```

The following commands:

```
.OPTIMIZE
.PARAMOPT M(NMOD1.3,VTH0)=(0.3, 0.1, 1)
```

and:

```
.OPTIMIZE
.PARAMOPT M(NMOD1.*,VTH0)=(0.3, 0.1, 1)
```

will change the parameter `VTH0` of the model `NMOD1.3` and all models `NMOD1.*` respectively.

Note

For this kind of parameter, the initial value is ignored.

Optional Initial Value

By default, the **.PARAMOPT** command requires an initial value for the parameter to be optimized. However, it is possible to omit the parameter INITIAL_VALUE, and take the value that is available in the netlist. This can be achieved in two ways:

- Globally, the option **PARAMOPT_NOINITIAL** can be specified to inform Eldo that initial values are not specified. The following statement must be used:

```
.PARAMOPT VARIABLE_NAME=(
+ LOWER_BOUND, UPPER_BOUND
+ [, INCREMENT])
```

- For a single parameter, the following can be specified, and the initial value will be obtained from the netlist.

```
.PARAMOPT VARIABLE_NAME=(
+ *,
+ LOWER_BOUND, UPPER_BOUND
+ [, INCREMENT])
```

Specification of Correlated Parameters

Element parameters can be correlated during the optimization process. The parameters are specified through the command:

```
.CORREL EXPR ELEMENT={EXPRESSION}
```

Where element can take the form of:

```
E(ELEMENT_NAME,ELEMENT_PARAMETER)
EM(ELEMENT_NAME,MODEL_PARAMETER)
M(MODEL_NAME,MODEL_PARAMETER)
PARAMETER_NAME
```

Caution

The meaning of **.CORREL EXPR** is completely different to **.CORREL** in a Monte Carlo analysis.

The character string EXPRESSION can contain references to **E**(), **EM**() and **M**() these quantities must be associated with the **.PARAMOPT** command. Therefore, in the following case, P2 will not be affected by P1:

```
C1 1 0 P1
R1 1 0 P2
.PARAMOPT P1=(1N, 0.1N, 10N)
.CORREL EXPR P2=E(C1,C)
```

In the following case, P2 will be affected by P1:

```
C1 1 0 1N
R1 1 0 P2
```

```
.PARAMOPT E(C1,C)=(1N, 0.1N, 10N)
.CORREL EXPR P2=E(C1,C)
```

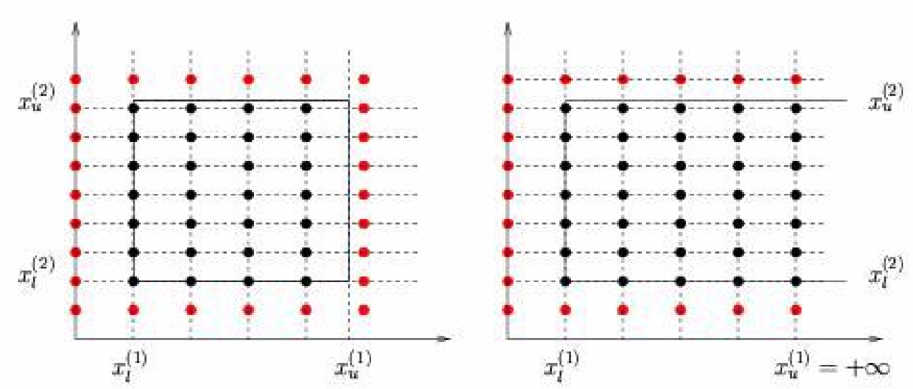
The purpose of the `.CORREL` command is to avoid editing files when an optimization is required to be performed on a non-parametric netlist. For example:

```
* LCR Parallel Network
VIN 1 0 AC 10R2 1 2 50
R3 2 3 50K
R5 3 0 50
L1 2 3 100U
C4 2 3 1N
VIN2 a 0 AC 10
RA A B 50
RB B C 50K
RC C 0 50
LA B C 100U
CA B C 1N

.PARAMOPT E(C4,C)=(1N,1N,20N)
.CORREL EXPR E(CA,C)='3* E(C4,C)/(2+1)'
```

Discretization of Design Variables

Figure 18-1 illustrates such a situation. The continuous box represents the feasible domain specified by the upper and lower bounds, and the black bullets are the feasible points where the final parameters are allowed to lie.

Figure 18-1. Discretization of final parameters



For example, when the lower bound is a finite number, the set of discretized points are as follows:

$$\{x_l^{(i)}, x_l^{(i)} + \delta_i, x_l^{(i)} + 2\delta_i, \ldots\}$$

Where $\delta_i > 0$ is the given increment. When the lower bound is infinite ($x_1^{(i)} = \infty$), the grid is started from the upper bound if this one is finite. The set of discretized points is then:

$$\{x_u^{(i)}, x_u^{(i)} - \delta_i, x_u^{(i)} - 2\delta_i, \ldots\}$$

When a design parameter is unbounded ($x_l^{(i)} = -\infty$ and $x_u^{(i)} = \infty$), the requirement of discretization is not considered.

Caution

In the Eldo optimizer, the obvious strategy of ignoring the requirement of discretization is used, solving the problem with real variables, and then rounding all the components to the nearest point onto the grid *at the end* of optimization. This strategy is not guaranteed to give solutions that are close to optimal. Increments on design parameters should only be used with design parameters for which rounding the optimized parameters is necessary. A detailed case is given in "Additional experiments" on page 1153.

Optimization Options

The following are a list of options that are used for optimization:

- **OPSELDO_ABSTRACT**

  Generates a summary table of simulations containing parameter and extract values for each run. This option has no effect on the *.otm* file, only the *.chi* file is affected.

- **OPSELDO_DETAIL**

  If this option is set to **NONE**, only the last run and the nominal run will be stored in the generated files (*.wdb*, *.aex*) and no other simulation information will be displayed on the standard output. When set to **ALL**, simulation information for all runs will be stored. Default is NONE. (This option was named **OPSELDO_NODETAIL** in pre-v6.6 Eldo versions and was not enabled.) This option only affects the standard output (which is controlled by Eldo).

- **OPSELDO_NETLIST**

  Generates a netlist modified from the original input file, which contains the optimized parameter values but also every parameter set under #ifdef statements.

- **OPSELDO_OUTPUT**

  This option controls the results of the optimization for the **DICHOTOMY**, **SECANT**, and **PASSFAIL** methods:

  OPSELDO_OUTPUT=0
  Prints results in a simplified format: parameter name, goal, optimized value

  OPSELDO_OUTPUT=1
  Prints results using the simplified format (as for **OPSELDO_OUTPUT**=0) and generate a *.ops0* file using the same format.

  OPSELDO_OUTPUT=2
  Prints results in a detailed format: parameter name, minimum value, maximum value, weight, goal, and optimized value

  OPSELDO_OUTPUT=3
  Prints results using the detailed format (as for **OPSELDO_OUTPUT**=2) and generate a *.ops0* file using the same format.

- **`OPSELDO_OUTER`**

  Allows a reverse behavior of optimization and sweep simulations (**`.TEMP`**, **`.DATA`**, or **`.STEP`**). A full optimization will be performed for each set of sweep parameters. Please refer to the section "Outer and inner design parameters" on page 1170. This option should be considered as obsolete, since the **`OUTER`** argument on the **`.OPTIMIZE`** command offers a better control over the outer parameters.

- **`PARAMOPT_NOINITIAL`**

  By default, the **`.PARAMOPT`** command requires an initial value for the parameter to be optimized. It is possible to omit the parameter `initial_value`, and take the value that is available in the netlist. Consider the following example (the full netlist is available in the BJT optimization example):

```
* THE MODEL
 .MODEL NPN   NPN (
 + XTF = XTF          VTF = VTF          ITF  = ITF
 + MJS = 0.00000      PTF = 0.00000      VJS  = 0.50000
 + CJS = 0.00000      BF  = 138.63       BR  = 1.7509
 + VAF = 63.743       VAR = 22.523       IS  = 3.25401E-16
 + IKF = 4.49066E-02  ISE = 3.11122E-16  NF  = 0.99595
 + NE  = 1.3692       IKR = 1.14501E-02  ISC = 1.48627E-15
 + NR  = 0.99737      NC  = 1.1209       RBM = 1.00000E-03
 + RB  = 294.75       RC  =  14.396      RE  = 4.1147
 + MJE = 0.20000      MJC = 0.20000      XCJC= 0.84276
 + TF  = TF           TR  = TR           VJE = 0.85188
 + VJC = 0.67113      FC  = 0.80607      IRB = 6.12980E-05
 + CJE = 2.48421E-12  CJC = 3.43125E-12 )

* DESIGN PARAMETERS
.PARAM
+ XTF=1 ITF=0.5 VTF=1 TF=3E-8 TR=1E-7
```

The statements defining the optimization variables can then be added to the netlist (or in an included file) with the following commands:

```
* OPTIMIZATION STATEMENTS
.OPTION PARAMOPT_NOINITIAL
.OPTIMIZE

.PARAMOPT
+ XTF = (0.1, 20)
+ ITF = (0.1,  5)
+ VTF = (0.1, 10)
+ TF  = ( 1P, 1U)
+ TR  = ( 1P, 1U)
```

Note

The command **.PARAMOPT** is an extension of the **.PARAM** command. It means that Eldo checks for the coherent definition of parameters. If a design parameter, such as **XTF**, is redefined with the statement **.PARAMOPT XTF**=(1,0.1,20), the warning message "Double definition for parameter XTF" will be generated. Use the option **NOWARN**=46 in order to disable these messages.

Basic Examples of Circuit Optimization

Table 18-3 gives a brief description of the basic examples used. These circuits are available in the directory: *$MGC_AMS_HOME/examples/optimizer/*

Table 18-3. Basic Examples of Circuit Optimization

Example	File name	Description
Low noise amplifier	*lnaopt.cir*	Optimization using AC, Noise and SST analyses
Fourband filter	*fourband.cir*	Optimization of a filter response
MOS characterization	*nmos.cir*	Double DC sweep optimization and ALTER blocks
Robust optimization using corners	*aop_optim.cir*	Optimization of a 2-stages operational Amplifier using ALTER blocks

Designing a Low Noise Amplifier (LNA)

The Low Noise Amplifier (LNA) architecture is a fully balanced dual-gain amplifier to achieve gain, linearity, and the noise specifications for a Zero-IF receiver architecture. Output power matching is not required since the output load is given by the integrated mixer.

This example will show how the Eldo optimizer can be used for a LNA. The architecture of the LNA is based on the 2.45GHz Switched-Gain CMOS LNA [6]. The high-gain stage is a fully balanced cascade configuration with an integrated LC tank as the load. This design reduces the Miller effect and improves isolation (-S12). The low-gain stage consists of two NMOS devices used as switches to achieve the required linearity and insertion loss.

Problem definition

The main characteristics are: Voltage Gain (AV), Input Power Matching (S11), Noise Figure (NF), and Input Third Order Intercept Point (IIP3). The design parameters are:

- Input Network Model (`CPIN1`, `CPIN2`, `LSIN`, `CSIN`)
- Source Inductor (associated with serial resistance)
- NMOS Width (composed of 10µm length fingers)

The input network consists of a serial capacitor (`CSIN`) used to isolate the LNA DC from the input. A π-network (consists of `CPIN1`, `CPIN2`, `LSIN`) enables simultaneous input power and noise matching. Changing the source inductor (`LS`) and the width of the NMOS through the number of fingers (`N1`) will improve noise matching, input power matching, and linearity characteristics such as IIP3 (the current and the load values are fixed to 6mA and 200Ω respectively).

Analyses

Using a single testbench, different analyses were simulated to extract the following key specifications:

- An AC analysis in order to extract Return Loss and the Voltage Gain.
- A NOISE analysis for extracting the Noise Figure.
- A multi-tone SST analysis for the IIP3.

```
* Parameters
.PARAM VG=0.6 VD=1.8
.PARAM FUND1=2.45G FUND2=2.46G PIN=-50
.PARAM IS=6m ROUT=200 RS=LS/1N

VIN  IN  0 RPORT=50 IPORT=1 AC 1 FOUR FUND1 FUND2 PDBM (1,0) PIN -90
+ (0,1) PIN -90
VOUT OUT 0 RPORT=ROUT IPORT=2

* Analyses
.DC
.AC LIN 11 2G 3G
.SST FUND1=FUND1 NHARM1=5 FUND2=FUND2 NHARM2=5
.NOISE V(OUT) VIN 10
.SNF INPUT=(VIN) OUTPUT=(VOUT)

* Plots
.PLOT AC SDB(1,1)
.PLOT NOISE DB(SNF) DB(NFMIN)
.DEFWAVE AV_DB=VDB(OUT)-VDB(IN)
.PLOT AC W(AV_DB)
```

The **`.DEFWAVE`** command was used to define the voltage gain (dB).

Design variables

For the optimization analysis, each parameter has an initial value, together with lower and upper bounds. For example, the capacitor `CPIN1` has an initial value of 0.1p with lower and upper bounds of 0.10p and 10p respectively. The variables `LS` and `N1` are specified using the fourth argument of the **`.PARAMOPT`** command. It means that these parameters are allowed to lie only on a discretized grid (the fourth parameter gives the step of this grid). Refer to "Discretization of Design Variables" on page 1146 for more information, and also the "Additional experiments" on page 1153. Our problem has bound constraints on the variables:

```
.PARAMOPT
+ CPIN1=(0.1P,0.10P,10P)        ! 0.1 × 10^-12 ≤ CPIN1 ≤ 10 × 10^-12
+ CPIN2=(0.1P,0.10P,10P)        ! 0.1 × 10^-12 ≤ CPIN2 ≤ 10 × 10^-12
+ LSIN=(0.1N,0,30N)             ! 0 ≤ LSIN ≤ 30 × 10^-9
+ CSIN=(1P,0.10P,10P)           ! 0.1 × 10^-12 ≤ CSIN ≤ 10 × 10^-12
+ LS=(0.25N,0,3N,0.25N)         ! 0 ≤ LS ≤ 3 × 10^-9
+ N1=(30,10,50,5)               ! 10 ≤ N1 ≤ 50
```

The degenerative integrated inductor (LS) would typically come from a design kit library. In our context, it is useful to specify the unit step (0.25n) to obtain the optimal matching inductor.

Design objectives and extracted measures

To specify design objectives, the keyword **GOAL** is used on the **.EXTRACT** command, this describes the ideal target that the user would like to reach for this specification with the optimization process.

```
* AC Analysis
.EXTRACT AC LABEL=AV_dB@2.4GHz YVAL(WR(AV_DB),2.4G)
+ GOAL=20

.EXTRACT AC LABEL=AV_dB@2.4GHz YVAL(WR(AV_DB),2.4G)
+ GOAL=20

.EXTRACT AC LABEL=S11_dB@2.4GHz YVAL(SDB(1,1),2.4G)
+ GOAL=-15

.EXTRACT AC LABEL=S11_dB@2.5GHz YVAL(SDB(1,1),2.5G)
+ GOAL=-15

* Noise Analysis
.EXTRACT NOISE LABEL=NF_dB@2.4GHz YVAL(DB(SNF),2.4G)
+ GOAL=MINIMIZE

.EXTRACT NOISE LABEL=NF_dB@2.5GHz YVAL(DB(SNF),2.5G)
+ GOAL=MINIMIZE

.EXTRACT NOISE LABEL=NFMIN_dB@2.4GHz YVAL(DB(NFMIN),2.4G)
+ GOAL=MINIMIZE

.EXTRACT NOISE LABEL=NFMIN_dB@2.5GHz YVAL(DB(NFMIN),2.5G)
+ GOAL=MINIMIZE
```

The specified design objectives finally lead to an optimization problem having four **GOAL** objectives and four **MINIMIZE** objectives.

The optimization process stops when the accuracy on the design variables need not be improved further, or more precisely, when the current point $\bar{x}$ satisfies the *optimality condition*. This optimality property can be checked in the Eldo optimizer output (*.otm f*ile):

```
Optimization Results
====================

Section 1 - Statistics and Diagnostics
--------------------------------------

 ==> Number Of Iterations     84
     Number Of Simulations    850

     Merit Function      4.8781332e+00
     Optimality               7.55e-03
     Feasibility              0.00e+00
```

The value labeled `Optimality` was denoted by `OPTIM(x)` in the definition of the **`.OPTIMIZE`** command. A small value indicates that the optimality condition is satisfied. The term `Merit Function` is the value of the function minimized during the optimization process. This function can be considered as a global score associated to the final iterate, i.e. it gives the distance of the final solution to the ideal solution (which may not exist). For the LNA problem, the ideal solution would have given a `Merit Function` value close to 4.0, since the four error functions coming from the objectives `AV_DB@2.4GHZ`, `AV_DB@2.5GHZ`, `S11_DB@2.4GHZ`, and `S11_DB@2.5GHZ` would be exactly zero, and the four **`MINIMIZE`** objectives should be close to 1.0.

The design parameters found by the optimizer provide a voltage gain close to 20dB, a return loss of 12dB, and an IIP3 of 1.2dBm. Note that the value of the Noise Figure (NF) is not so similar to the value of the Minimum Noise Figure (NFMIN). Thus the noise matching can be improved, however, this maybe at the expense of S11 matching. The user is invited to use different weighting numbers for each of the objectives. Table 18-4 gives the results of optimization for the **`GOAL`** objectives:

Table 18-4. Results of Optimization for GOAL Objectives

Name	Initial Value	Current Value	Goal Value
`AV_DB@2.4GHZ`	11.9	20.1	20.0
`AV_DB@2.5GHZ`	11.7	19.6	20.0
`S11_DB@2.4GHZ`	-0.82	-15.1	-15.0
`S11_DB@2.5GHZ`	-0.88	-15.1	-15.0

The results reported in the previous table have been obtained with the default values (1.0) of the **WEIGHT** arguments. The results for **MINIMIZE** objectives are shown in Table 18-5:

Table 18-5. Results of Optimization for MINIMIZE Objectives

Name	Initial Value	Current Value
NF_DB@2.4GHZ	3.03	1.41
NF_DB@2.5GHZ	3.06	1.41
NFMIN_DB@2.4GHZ	1.01	0.97
NFMIN_DB@2.5GHZ	1.06	1.03

Additional experiments

As with many problems in circuit design, we noted above that some of the variables are restricted to be members of a finite set of values. These variables were LS, the NMOS source inductor, and N1, the number of fingers in the NMOS. It is interesting to use the LNA problem to illustrate the distinction between the two types of discrete variables that you may encounter in practical problems. Some of the techniques exposed in this section are freely adapted from the book of Gill, Murray and Wright [4].

First consider the LNA problem with respect to the variable LS: we shall assume the N1 variable fixed at its initial value N1=30. The problem appears to be an optimization problem with continuous variables. What makes it a mixed continuous-discrete problem is the fact that the LS inductor comes from a design kit library which restricts the values to specific sizes. Such variables are termed *pseudo-discrete*, since the solution to the continuous problem (in which the variables are not subject to discrete restrictions) is well defined.

The other category of discrete variables is represented by the variable N1. It is clearly one for which there is no physical meaning of a non-integer value. In our case, what makes the LNA problem related to a problem with continuous variables, is the definition of the NMOS model parameters used in the design kit library. For example, the parameter WR, which represents the width offset for the channel resistance (RDS) calculation, is redefined as follows:

```
.PARAM WR='(C1 + C2*Lm - C3*N1)'
```

The symbols C1, C2, and C3 refer to some constant factors which we do not specify here. The parameter Lm is simply the length of the transistor. This definition of the WR parameter does not use the discrete nature of the N1 parameter. During the evaluation of this expression, the value of N1 is considered as a floating number. It means that the problem is implicitly re-formulated as continuous. Note that the situation is more difficult to treat when the integer part of N1 is used explicitly in the previous expression:

```
WR='(C1 + C2*Lm - C3*TRUNC(N1))'
```

Such situations should be avoided as much as possible by designers.

We will now illustrate how our problem with pseudo-discrete variables can be solved by utilizing the solution of the continuous problem. The results of the optimization are summarized in the *.otm* file, and the information shown in Table 18-6 can be extracted:

Table 18-6. Extracted Information from .otm File

Name	Initial Value	Final Value	Discretized value	Increment
LS	2.5×10^{-10}	1.5352×10^{-9}	3.0×10^{-9}	1.5×10^{-9}
CPIN1	1.0×10^{-13}	0.0	0.0	0.0
CPIN2	1.0×10^{-13}	8.7506×10^{-14}	2.2340×10^{-13}	0.0
CSIN	1.0×10^{-12}	1.0×10^{-12}	1.0×10^{-12}	0.0
LSIN	1.0×10^{-10}	1.3651×10^{-8}	1.3651×10^{-8}	0.0
N1	30	18.453	20	5

The optimal N1 value when treated as a *continuous* variable is 18.453, and we may suppose the *optimal discrete* value is unlikely to be very different. This will be confirmed by the next two experiments. The first experiment represents a general approach for treating optimization problems with pseudo-discrete variables, while the second illustrates what we call a “well-defined” problem.

We defined `N1` as the variable that must assume one of the integer values {`10`, `15`, `20`, ..., `50`}. Let $N1^C$ denote the value of `N1` at the solution of the continuous problem, which is supposed to be unique. Suppose that $N1^C$ satisfies:

$$n_s < N1^C < n_{s+1}$$

The value of the merit function `Fmerit` at the continuous solution is a lower bound on the value of `Fmerit` at any solution of the discrete problem, since if the variable N1 is restricted to be any value other than $N1^C$, the merit function for such a value must be larger than `Fmerit` at the continuous solution, irrespective of the other variables (`LS`, `CPIN1`, `CPIN2`, ...).

The next stage of this process is to fix the pseudo-discrete variables at either ns or ns_{+1}. This choice is achieved by combining the values in Table 18-7:

Table 18-7. Combinations of N1 and LS Values

Name	s	s+1
N1	15	20
LS	1.5×10^{-9}	1.75×10^{-9}

The problem is then solved again, minimizing with respect to the remaining continuous variables, using the old optimal values as the initial estimate of the new solution. As shown in Table 18-8, solving the restricted problem should require only a fraction of the effort needed to solve the continuous problem. This is because the number of variables is smaller, and the solution of the restricted problem should be close to the solution of the continuous problem.

Table 18-8. Results of Restricted Problem

Run	F_{merit}	Nb Iter	Nb Simul
`N1=20, LS=`1.5×10^{-9}	4.9403493	16	130
`N1=20, LS=`1.75×10^{-9}	5.4893925	16	131
`N1=15, LS=`1.5×10^{-9}	5.4792729	57	467
`N1=15, LS=`1.75×10^{-9}	5.0082957	40	329

The value of the merit function `Fmerit` at the continuous solution is a lower bound on the restricted solutions. This value was given above and is 4.8781332. The solution obtained for the run with `N1=20` and `LS=` 1.5×10^{-9} will be a satisfactory solution. The extra computation cost in solving the additional restricted problems associated with the discrete variables is likely to be a fraction of the cost to solve the original full continuous problem. If not, this implies that the discrete solution differs substantially from the continuous solution.

The following experiment can be considered as a complete enumeration procedure, where all the possible combinations of the discrete values of `N1` and `LS` were tested. It is important to note that this kind of process is very costly and inefficient. It is given for the purpose of illustration.

The plot shown in Figure 18-2 has been obtained by extracting the optimal value of the merit function F_{merit} after solving each problem associated to a restricted problem with fixed values `N1` and `LS`. The following statements were used:

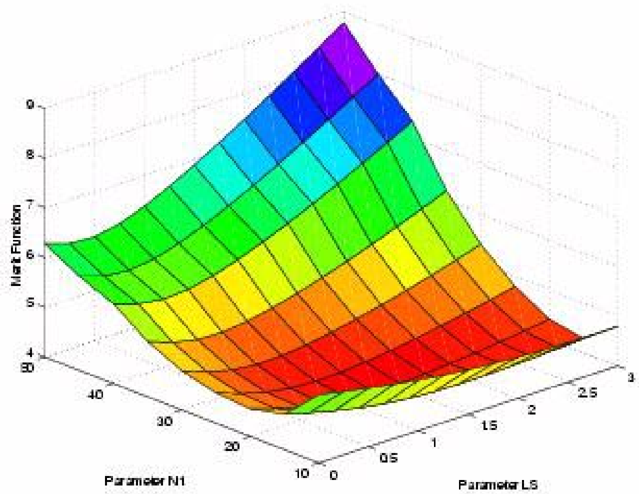
```
.PARAM LS=0.25n
.PARAM N1=30

.STEP PARAM LS 0 3n 0.25n
.STEP PARAM N1 10 50 5

.OPTIMIZE OUTER=LS,N1
```

This information is available as follows at the end of the *.otm* file (some columns were removed in order to simplify the results):

```
Optimization Results With Respect To The Outer Loop
===================================================

  Run   Iter  Simul   Merit Function Optim Outer Parameters

     1    22   177     5.0741859e+01 6.4e-04 LS: 0.0000e+00  N1: 1.0000e+01
     2    11    97     6.7559608e+01 1.5e-03 LS: 0.0000e+00  N1: 1.5000e+01
     3    26   205     6.2095585e+01 2.3e-04 LS: 0.0000e+00  N1: 2.0000e+01
     4    33   274     5.2472654e+01 1.1e-01 LS: 0.0000e+00  N1: 2.5000e+01
     5    31   242     4.1993472e+01 9.3e-02 LS: 0.0000e+00  N1: 3.0000e+01
...
```

The resulting plot of the merit function shows that there is a narrow area along the line `N1=20` where this function takes its minimal values. This means that the assumption of a well-defined problem having continuous and pseudo-discrete variables is reasonable.

Figure 18-2. Values of the merit function

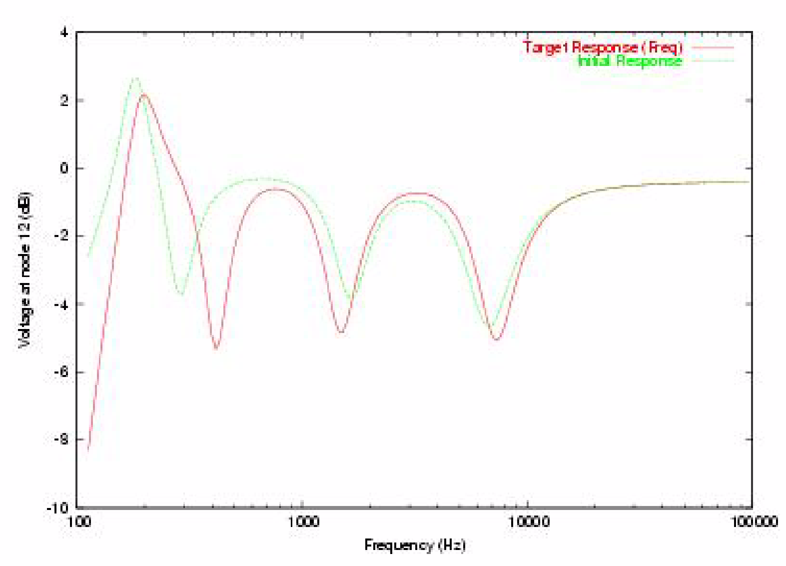


Fourband Filter Optimization

This simple example (which first appeared in the former optimization software Opsim) illustrates the fitting of a filter response.

This problem consists of the optimization of a set of eighteen circuit parameters such that the voltage at node 12 follows the curve response given in Figure 18-3. The initial response is also plotted.

Figure 18-3. Response of the filter



Design variables

The design variables are some resistor and capacitor values of the circuit. The initial (or nominal) values are given below. The capacitors `CT`, `CLC`, and `CPASS` have been specified separately in order to illustrate the effect of the initial conditions on the optimizer. Please refer to the section "Global and Local Optimization" on page 1176 for more information.

```
.PARAM
+ R2=10K CL=0.01U R2B=8K
+ CTB=0.01U CLB=0.01U R2C=12K CTC=0.002U
+ R2D=12K CTD=800P CLD=150P
+ CACT1=0.02U CACT2=0.02U
+ RPASS=4K RACT1=50K RACT2=180K

.PARAM CT=0.02U CLC=0.001U CPASS=0.3U
```

with the associated **.PARAMOPT** statements:

```
.PARAMOPT
+ R2    = (    5K,    20K)
+ CT    = (0.001U,  0.05U)
+ CL    = (0.001U,  0.05U)
+ R2B   = (    3K,    10K)
+ CTB   = (0.001U,  0.05U)
+ CLB   = (0.001U,  0.05U)
+ R2C   = (    5K,    20K)
+ CTC   = (0.001U,  0.01U)
+ CLC   = (0.001U, 0.005U)
+ R2D   = (    5K,    20K)
+ CTD   = (  100P,  1000P)
+ CLD   = (   10P,   500P)
+ CPASS = (  0.1U,   1.0U)
+ CACT1 = (0.001U,  0.05U)
```

```
+ CACT2 = (0.001U,  0.05U)
+ RPASS = (    1K,     5K)
+ RACT1 = (   20K,   100K)
```

Analyses

A single AC analysis is performed, this is driven by **.DATA** command.

```
.AC DATA=DATA_FOURBAND

.DATA DATA_FOURBAND
+    FREQ          VDB12
+  1.122E+02    -8.326E+00
+  1.166E+02    -7.435E+00
+  1.212E+02    -6.566E+00
...
+  8.913E+04    -4.212E-01
+  9.261E+04    -4.213E-01
+  9.624E+04    -4.215E-01
.ENDDATA
```

Design objectives

The optimization will be performed on a single design objective to improve the match of the constructed model (or the target curve) with the actual measurements. The differences between the model and the measured values are combined in a global error function (using the least squares approach). The objective statement is as follows:

```
.OBJECTIVE AC LABEL=RESPVDB12
+ VDB(12) GOAL=DATA_FOURBAND(VDB12)
```

Former releases of the Eldo optimizer used the equivalent (but less explicit) **FITTING** construct:

```
.EXTRACT AC LABEL=RESPVDB12 FITTING
+ DATA_FOURBAND(VDB12) VDB(12)
```

Optimization results

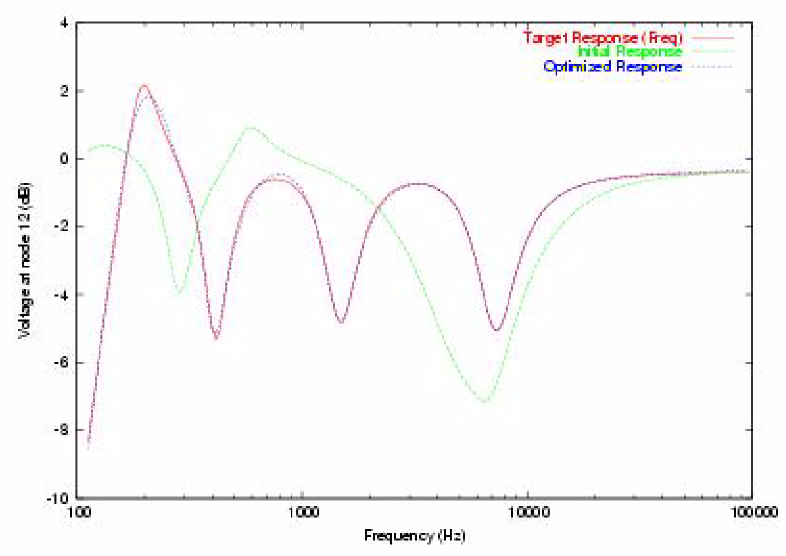
The values shown in Table 18-9 can be found in the file *fourband.otm* generated during the optimization process. The optimization was performed on a 32-bit Linux machine.

Table 18-9. Values Extracted from fourband.otm File

	Minimum Error	Global Error	Maximum Error
Test 1	1.7×10^{-6}	1.6×10^{-6}	7.3×10^{-3}
Test 2	1.4×10^{-4}	1.2×10^{-2}	4.3×10^{-1}

The minimum and maximum errors give the range of the residuals (difference between the target and the measured values) after optimization. The global error represents the error function minimized by the optimizer (this value may differ with the merit function of the optimizer). The optimized response is not plotted in Figure 18-3 since the target and the output responses are the same.

Figure 18-4. Optimized response (starting from a different point)



Now consider the following initial values of the capacitors `CT`, `CLC`, and `CPASS`:

```
.PARAM CT=0.002U CLC=0.0001U CPASS=1.0U
```

The corresponding response is plotted in Figure 18-4 and the minimum and maximum errors are stored in previous table.

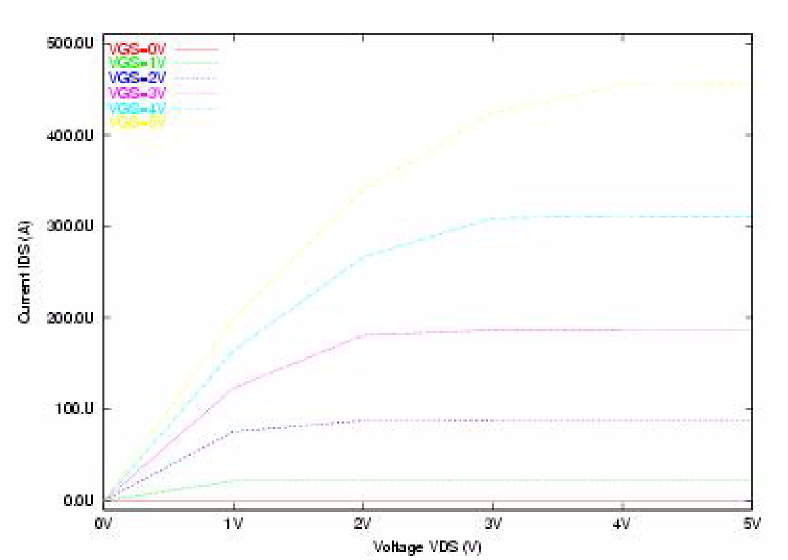
This example illustrates the influence of the initial point in circuit optimization and the relative robustness of the optimizer. The second test is much more difficult to solve.

MOS Characterization

This example illustrates the combination of a double `DC` sweep and `ALTER` blocks for the optimization of I-V data for a Level 3 MOS model. The data consists of drain characteristics (IDS versus VDS and VGS) with two different choices of the length parameter `L`.

The data is stored in two separate **`.DATA`** statements (*d25x25.dat* and *d25x2p5.dat* files). The I-V curves are shown in Figure 18-5 at the initial point of optimization.

Figure 18-5. Illustration of I-V characteristics



Circuit statements

The circuit and the model statement are given below. The circuit is a simple single transistor circuit. The parameter VDS, VGS and VBS are the names of the sweep parameters used in the parametric DC analyses.

```
*   CIRCUIT STATEMENTS
VDS        104    0     VDS
VGS        102    0     VGS
VBS        103    0     VBS
M25X25D1   101    102   0   103   NMOS W=25U L=25U NRS=0.12 NRD=0.12
V101       104    101   0

*   THE MODEL
.MODEL NMOS NMOS(LEVEL=3 TOX=2.0E-8 NSUB=1E16
+ LD=LD VTO=VTO GAMMA=GAMMA UO=UO VMAX=VMAX
+ THETA=THETA ETA=ETA KAPPA=KAPPA)
```

ALTER blocks and design objectives

In our example the netlist consists of two blocks, the second is defined with a **.ALTER** command. It is important to understand that the first **.OBJECTIVE** command has a local scope. This means that the expression I(V101) is computed in a specific simulation context. This context is related to a ALTER block (the first block may be considered as the root block or nominal case).

```
* DRAIN CHARAC  W=25U L=25U AT VBS=0.0
.INCLUDE d25x25.dat
.DC DATA=DATA_D25X25 ! DATA DRIVEN ANALYSIS

.OBJECTIVE DC LABEL=FIT_IDRAIN I(V101) ! DOUBLE DC SWEEP OPTIMIZATION
+ GOAL=DATA_D25X25(IDS) TYPVAL=1E-4
```

```
* DRAIN CHARAC  W=25U L=2.5U AT VBS=0.0
.ALTER DRAIN 25X2P5 AT VBS=0
M25X25D1 101 102 0 103 NMOS W=25U L=2.5U NRS=0.12 NRD=0.12

.INCLUDE d25x2p5.dat
.DC DATA=DATA_D25X2P5

.OBJECTIVE DC LABEL=FIT_IDRAIN I(V101)
+ GOAL=DATA_D25X2P5(IDS) TYPVAL=1E-3
```

The argument **TYPAL** has been used to globally rescale the objectives (the extracted measures and the associated target values), such that the low current data is not dominated by others data.

Design variables

This example performs optimization of the level 3 parameters: LD, VTO, GAMMA, UO, VMAX, THETA, ETA, and KAPPA. The corresponding statements are:

```
.PARAM
+ LD=1E-7 VTO=1.0 GAMMA=0.6 UO=600 VMAX=1E5
+ THETA=0.1 ETA=0.01 KAPPA=1

.OPTION PARAMOPT_NOINITIAL

.PARAMOPT
+ LD    = ( 1E-9, 5E-7)
+ VTO   = (    0,    2)
+ GAMMA = (  0.1,    2)
+ UO    = (  300,  900)
+ VMAX  = (  1E4,  1E7)
+ THETA = ( 1E-6,    2)
+ ETA   = ( 1E-4,    2)
+ KAPPA = ( 1E-6,  1E3)
```

Optimization results

The values presented in Table 18-10 can be found in the *nmos.otm* file generated during the optimization process. The optimization was performed on a 32-bit Linux machine.

Table 18-10. Values Extracted from nmos.otm File

	Minimum Error	Global Error	Maximum Error
Length L=25U	0.0	1.1×10^{-18}	3.5×10^{-9}
Length L=2.5U	0.0	6.4×10^{-17}	3.0×10^{-8}

Note that these values are the result of a synthetic problem. The circuit parameters were fixed, the data was then generated, and the circuit was optimized after performing a perturbation of the initial point. This is why the global error is so small. The residuals are almost zero. This is not the case with realistic data where the physical measurements involve some variability in the current curves.

Robust Optimization Using Corners

This example illustrates the combination of optimization and **.ALTER** commands. For this purpose a new architecture based on the concepts of multi-context of simulation and multi-netlist optimization has been developed.

Problem definition

Nominal optimization focuses on finding the *best* design parameters for one *nominal* operating condition of the circuit. Typically the power supply, the ambient temperature, and the process technology are given their nominal value, and the best set of design parameters is found by the optimizer, given its targets. If the circuit has to operate under a variety of operating conditions, nothing guarantees that this will still be the case, if the design parameters are simply set to these optimal nominal values. Maybe if the power supply level is slightly changed, the circuit will fail.

'Robust optimization' focuses on finding the 'best' set of parameters that fulfill the specifications across a certain range of operating conditions. For example, robust optimization would be performed on a circuit that must operate between 1.7 and 1.9V, a temperature range of -25C to 100C, and accommodate variations in the process (defined by the 'corner' device model libraries). In this case the optimization targets might be upper or lower bounds on certain characteristics (for example the DC consumption has to be 'lower than 50μA, in all operating conditions). Targets can be set as targets for the average value of a given specification.

Operating conditions are conveniently defined with **.ALTER** commands in Eldo. In each **.ALTER** command, a specific combination of parameters defining the operating conditions (typically the power supply level and the temperature) and the corner device model library, can be defined. The optimization commands (design variables definition, design objective definitions, and the optimize command) from the main netlist are then interpreted to span the main netlist conditions *and* the various combinations defined through the **.ALTER** sections. Obviously these types of optimizations are usually more costly than simple nominal optimizations. Eldo can distribute the necessary simulation on multi-processor machines, thus potentially accelerating the process.

Circuit statements

The following netlist is available in the directory *$MGC_AMS_HOME/examples/optimizer*

```
* 2-STAGES OPAMP - TYPICAL MODEL FOR 3.3V

M16 VDD  BIAS M17D VDD PCH_33  W='WS*4.8U' L=0.8U
M17 M17D M17D VSS  VSS NCH_33  W='WS*3.2U' L=0.8U
M15 M15D M17D VSS  VSS NCH_33  W='WS*3.2U' L=0.8U
M13 M13D M17D VSS  VSS NCH_33  W='WS*3.2U' L=0.8U
M14 VDD  M15D M15D VDD PCH_33  W='WS*1U'   L=0.8U
M12 VDD  M13D M13D VDD PCH_33  W='WS*4.8U' L=0.8U
M3  VDD  M13D M1D  VDD PCH_33  W='WS*4.8U' L=0.8U
M4  VDD  M13D M2D  VDD PCH_33  W='WS*4.8U' L=0.8U
```

```
M11 VDD  M13D OUT  VDD PCH_33  W='WS*16U'  L=0.8U
M1  M1D  INP  COM  COM NCH_33  W='WS*0.8U' L=0.6U
M2  M2D  INN  COM  COM NCH_33  W='WS*0.8U' L=0.6U
M9  COM  M17D VSS  VSS NCH_33  W='WS*3.2U' L=0.8U
M5  M2D  M15D M7D  VDD PCH_33  W='WS*2.4U' L=0.8U
M6  M1D  M15D M8D  VDD PCH_33  W='WS*2.4U' L=0.8U
M7  M7D  M7D  VSS  VSS NCH_33  W='WS*1.6U' L=0.8U
M8  M8D  M7D  VSS  VSS NCH_33  W='WS*1.6U' L=0.8U
CX  M8D  OUT  C_COMP ! LOUSY COMPENSATION CAP.
CL  OUT  0    1P     ! LOAD CAP.
M10 OUT  M8D  VSS VSS NCH_33  W='WS*13U' L=0.8U

.CONNECT OUT INN ! FOLLOWER CONNECTION

.OP

.TRAN 1N 400N
.PLOT TRAN V(INP) V(OUT)

.OPTION TUNING=VHIGH NOASCII NOMOD

VDD VDD 0  1.65V
VSS VSS 0 -1.65V
VINP INP 0 PULSE -0.9 0.9 10N 1N 1N 200N 400N AC 1 0
VB BIAS 0 0.55V

* TYPICAL MODEL FOR 3.3V DEVICES
.LIB ./cln90g_lk.eldo TT_33
```

.ALTER sections

```
The following corners are used:
* SS_33 : Slow NMOS Slow PMOS model for 3.3V devices
.ALTER SS_33 : SLOW NMOS SLOW PMOS MODEL
.LIB ./cln90g_lk.eldo SS_33

* FF_33 : Fast NMOS Fast PMOS model for 3.3V devices
.ALTER FF_33 : FAST NMOS FAST PMOS MODEL
.lib ./cln90g_lk.eldo FF_33

* SF_33 : Slow NMOS Fast PMOS model for 3.3V devices
.ALTER FS_33 : FAST NMOS SLOW PMOS MODEL
.LIB ./cln90g_lk.eldo FS_33

* FS_33 : Fast NMOS Slow PMOS model for 3.3V devices
.ALTER FS_33 : SLOW NMOS FAST PMOS MODEL
.LIB ./cln90g_lk.eldo SF_33
```

Analyses and design objectives

Two analyses were specified to define the following design objectives:

- A transient analysis in order to extract the output voltage.
- A DC analysis for extracting the IDS current.

```
.EXTRACT TRAN LABEL=RSLOPE SLOPE(V(OUT),0,0,140N)
+ GOAL=80.0E6 WEIGHT=1000.0

.PARAM SCAL_I=1E4 ! INVERSE OF TYPICAL VALUE
.EXTRACT DC   LABEL=SCAL_IBIAS SCAL_I*ID(M16)
+ GOAL=MINIMIZE WEIGHT=1.0
```

The constant parameter `SCAL_I` was used in order to rescale the extracted measure `IBIAS`. The values taken by `RSLOPE` and `SCAL_IBIAS` have the same order of magnitude. The specified weight for the measure `RSLOPE` was introduced to find a solution that minimize the rise time at the expense of larger `IBIAS` values. Users can experiment with different choices of weights.

Design variables

A shrink parameter `WS` is used for the width of each MOS instantiated in the netlist. The capacitor is also optimized:

```
.PARAMOPT
+ WS=(2,0.5,4)
+ C_COMP=(1p,0.01p,100p)
```

Optimization results

This example must be run with the following commands:

```
.OPTIMIZE TOL_OPT=1E-2
.OPTION OPSELDO_ALTER
```

where the option **`OPSELDO_ALTER`** informs Eldo that optimization must be performed on all **`.ALTER`** sections. The results of the optimization are placed in the optimization file with the extension *.otm*.

Set-up Time Computation

The goal here is to use Eldo to compute the set-up time of a flip-flop. The definition of set-up time is: time between input and the clock (`TIC`) so that propagation time between clock and output (`TCO`) is 10% above nominal value (computed when there is a large time between input and the clock).

To do this, suppose that the nominal value of `TCO` is already known.

$$\text{TCO\_TARGET} = 1.1 \times \text{TCO\_NOMINALS}$$

A very general method, available even with older versions of Eldo, would be to sweep `TIC`, measure `TCO`, and extract from `TCO(TIC)` the value of `TIC` so that `TCO=TCO_TARGET`.

```
.PARAM TIC=200n
.STEP PARAM TIC 205N 195N 0.1N
.EXTRACT LABEL=TCO
+ (XUP(V(Q),0.48,200n,300n,1) - XUP(V(CP),0.48,200n,300n,1))
```

```
.EXTRACT SWEEP xycond(XAXIS, meas(TCO)=1.1*TCO_NOMINAL)
```

Simulation time can be improved with the use of **.OPTION AUTOSTOP**=2. Accuracy is proportional to the parameter sweep increment, and simulation time is proportional to the number of steps in the swept interval. From Eldo v5.8, it is possible to use two new methods of optimization that are suitable for the computation of set-up the time. The **PASSFAIL** method computes the maximum value for which an extract can be measured. This is not exactly the definition of set-up, however, it can be used as a first step, this would give:

```
.OPTIMIZE METHOD=PASSFAIL RESULTS=TCO
.PARAMOPT TIC=(201N,199N,205N)
```

The **DICHOTOMY** method, finds the value of an optimization parameter so that one extract yields a target value, provided that the extract values for the `Min` and `Max` values of the optimization parameter bracket the target.

In this example, the `Min` value of the optimization parameter `TIC` has been computed with the **PASSFAIL** method, this will give:

```
.EXTRACT TRAN LABEL=DCP
+ (XUP(V(Q),0.48,200N,300N,1) - XUP(V(CP),0.48,200N,300N,1))
+ GOAL={1.1*TCO_NOMINAL}

.PARAMOPT TIC=(201N,200.07N,205N) !MIN provided by PASSFAIL
```

From Eldo v5.9, it is possible to combine both **PASSFAIL** and **DICHOTOMY** methods in one step with the following syntax:

```
.OPTIMIZE METHOD=DICHOTOMY
.EXTRACT tran LABEL=DCP
+ (XUP(V(Q),0.48,200N,300N,1) - XUP(V(CP),0.48,200N,300N,1))
+ GOAL={1.1*TCO_NOMINAL}
.PARAMOPT TIC=(201N,195N,205N)
```

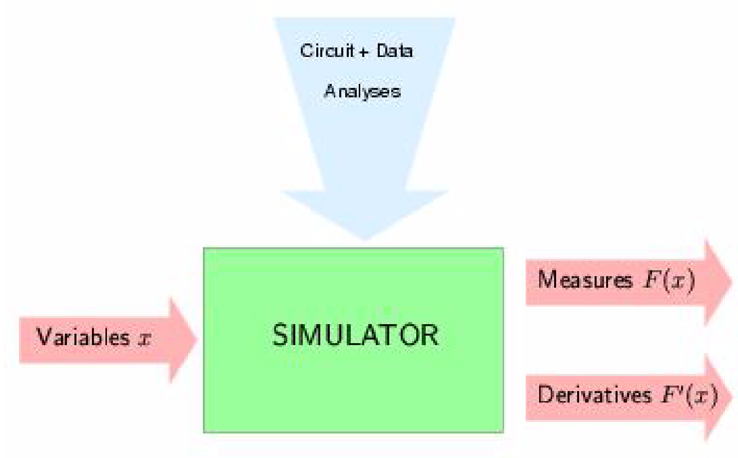
The **SECANT** method search algorithm is more advanced than the **DICHOTOMY** method. Many problems suitable for bisection (one design parameter, one target) are actually relatively linear, i.e. the target value is almost proportional to the design parameter. In these conditions, using the **DICHOTOMY** method is clearly sub-optimal in terms of the number of iterations required. Using the **SECANT** method will greatly diminish the number of iterations. This feature is useful for designers that want to accelerate their setup/hold simulations.

Modeling Capabilities in Eldo and Optimization

The optimization functionality implemented in the Eldo optimizer are designed to match as much as possible the modeling capabilities available through the Eldo language. As described in the introduction, the designer's request for optimization, the definition of the parameters to be optimized, the design objectives that they should satisfy are specified in the netlist file, by using simple extensions of the Eldo commands.

The simulator is regarded as a black box that provides a vector of measures F(x) given the set of design variables x, shown in Figure 18-6.

Figure 18-6. Simulator as black box



In general, the choice of the optimization parameters does not lead to severe difficulties. It is part of the designer's knowledge about the circuit, while it is not necessarily the case with the definition of the design objectives to be optimized (the function F(x)). Here is a list of issues associated to the question of the appropriate model for optimization:

- What kind of problems can be solved using the Eldo language constructs such as multiple sweep commands or step increments on parameters? How are `ALTER` blocks handled in the context of optimization? These groups of issues are related to the mathematical formulation of the optimization problem.
- What are the common issues related to the built-in functions used in the extraction of measures? For instance, a difficult situation arises when the design objectives are not continuous functions and/or do not have continuous derivatives.
- The mathematical issues that are peripheral in the problem of circuit design have to be considered, but essential to the computer solution of actual problems. That is how to adjust for problems that are badly scaled in the sense that the optimization variables or the design objectives are of widely differing magnitudes. It is also necessary to consider how to determine when to stop the iterative algorithms (used in the Eldo optimizer) in finite-precision arithmetic.

When optimizing a circuit, the mathematical aspects are not great concern. The user can avoid technical details, accepting when some explanations are required for better use of these optimization features.

What Problems can be Solved?

Representation of design objectives

This section is organized as follows:

- The notions of *design parameters* and *design variables*. These concepts are formal, but necessary in order to get a good view of the modeling capabilities.
- How to perform a sequence of optimizations while sweeping one or more circuit parameters.
- The specifications of design variables and design objectives are explained in detail.

Notions of design variables and design parameters

It is useful to consider circuit parameters separately that are the candidate for optimization (such as width and length for MOS) and those associated to DC multi-point, transient, small signal, frequency, or parametric analysis. The former ones will be denoted as *design variables* and the latter as *design parameters* (or sweep parameters).

Design variables are only specified through the **.PARAMOPT** command. They represent the vector of variables *x*. Optimization will always occur with respect to these variables. Conversely, the design parameters represent the values of a user-defined discretization of a real-valued set Ω. Those parameters are denoted by the Greek letter ω.

For example, the Voltage-Temperature parameters temperature-voltage (`TEMP`,`PVDD`):

```
.STEP TEMP -25 150 5
.STEP PARAM PVDD 2.7 3.3 0.1
```

define a cartesian product or a box $\Omega = [-25, 150] \times [2.7, 3.3]$ in the 2-dimensional space $\Re^2$. The design parameters have various origins, however, they are usually defined through sweep simulations. The way the box Ω can be discretized follows the capabilities offered by the using **.TEMP**, **.DATA** and **.STEP** commands.

The following characterization can shed some light on the role of x and ω:

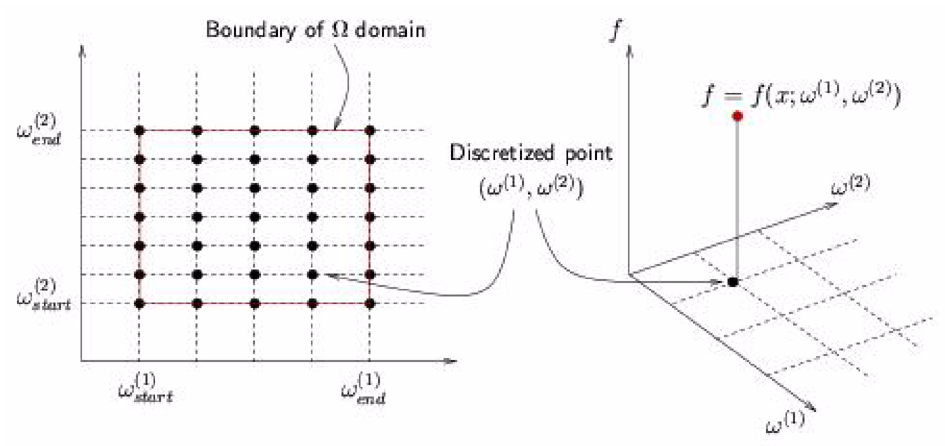
- the task of evaluating the extracted measures remains in the hands of the simulator, and the optimizer is only required to provide a new approximate solution x.
- It means that the parameters ω are only varying within the simulator, while the design variables are only affected by the optimization algorithm. The simulation and the optimization processes are *totally* distinct.

The example of a 2-dimensional domain Ω can be considered, as shown in Figure 18-7, and obtained from a multi-step specification:

```
* Design parameters
.STEP PARAM OMEGA_1 <INCR_SPEC_1>
.STEP PARAM OMEGA_2 <INCR_SPEC_2>

* Minimization of the design objective: f
.OBJECTIVE
+ EXTRACT_INFO LABEL=F
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE
```

The argument **GOAL**=**MINIMIZE** specifies that the extracted measure f must be minimized with respect to the variables x.

Figure 18-7. Discretization of design parameters



Due to the presence of the multiple increments on ω, the extracted measure f is a bi-dimensional object of size defined by the range of the **.STEP** statements. Given the values of the variables x (fixed during one single simulation), the simulator returns a value f= $f(x;\omega^{(1)}, \omega^{(2)})$ for each discretized point $(\omega^{(1)}, \omega^{(2)})$.

The argument **GOAL**=**MINIMIZE** on the extracted measure finally means that the whole set of values of f must be minimized with respect to x.

Explicit and implicit design parameters

When optimizing a circuit, the category of design parameters can be divided into *explicit* and *implicit* parameters. For the transient and the small-signal analyses the time and frequency, respectively, are clearly implicit parameters.

To specify a quiescent analysis (that is a component analysis CNAM), a voltage or current analysis (SNAM), a temperature analysis, and a parameter analysis the following can be specified:

```
.DC CNAM [L|W] START STOP INCR
.DC SNAM START STOP INCR [SNAM2 START2 STOP2 INCR2]
.DC TEMP START STOP INCR
.DC PARAM START STOP INCR
```

the parameters are always considered as implicit. In the context of optimization, Eldo will run the specified analyses with their nominal values (defined in the netlist).

An important exception is the **DC** analysis that is driven by a **.DATA** construct:

```
.DC DATA=DATA_SWEEP
```

Hierarchy and depth of design parameters

The modeling capabilities of Eldo allow various specifications of design parameters. In the following the hierarchy (or the default order of precedence) of the commands that define the variation of the parameters are given:

- Several **.ALTER** blocks with the label denoted by A_1, A_2, ...
- A **.TEMP** command or a **.STEP** command on temperature

```
.TEMP T1 T2 ...
.STEP TEMP T_START T_STOP T_INCR
```

- A **.STEP** command using different variants of libraries (or corners K):

```
.LIB mylib TYP
.STEP LIB(mylib) K1 K2 ...
```

- A **.STEP** command on circuit parameters

```
.STEP PARAM ALPHA ALPHA_START ALPHA_STOP ALPHA_INCR
```

- Multiple sweeps or nested sweeps on parameters, and a parameter sweep in the analysis command. These parameters will be denoted by α, β, ... The cases of **DC**, **AC**, and **TRAN** analyses are represented by the command syntax:

```
.{AC | TRAN} <SIMUL_POINTS> SWEEP DATA=DATA_SWEEP
.DC <SOURCES/PARAMETERS/...> SWEEP DATA=DATA_SWEEP
```

where a two parameter sweep is defined with:

```
.DATA DATA_SWEEP ALPHA BETA
+ ALPHA1 BETA1
+ ALPHA2 BETA1
...
+ ALPHAM BETA1
+ ALPHA1 BETA2
...
+ ALPHAM BETA2
...
+ ALPHAM BETAP
.ENDDATA
```

- A DC data-driven analysis:

```
.DC DATA=DATA_SWEEP
```

- Transient and frequency analyses. The time and frequency parameters will be denoted by the symbol ω.

It follows that a design objective F can be represented as a family of real-valued functions of variables x. This family is parameterized with points that were denoted by A, T, K, α, β,..., ω.

The following convention is used: the parameter omega is the "inner-most" and the **.ALTER** label A is the "outer-most". This convention tries to fit the order of simulations. The simulator provides to the optimizer this hierarchy for specifying the depth of a design parameter. This is used in printing routines and optimization algorithms. It is possible to change the depth of a design parameter when it is necessary, please refer to "Outer and inner design parameters" on page 1170 for more information. This feature is limited to the parameters defined in a **.STEP PARAM** or **.STEP TEMP** command.

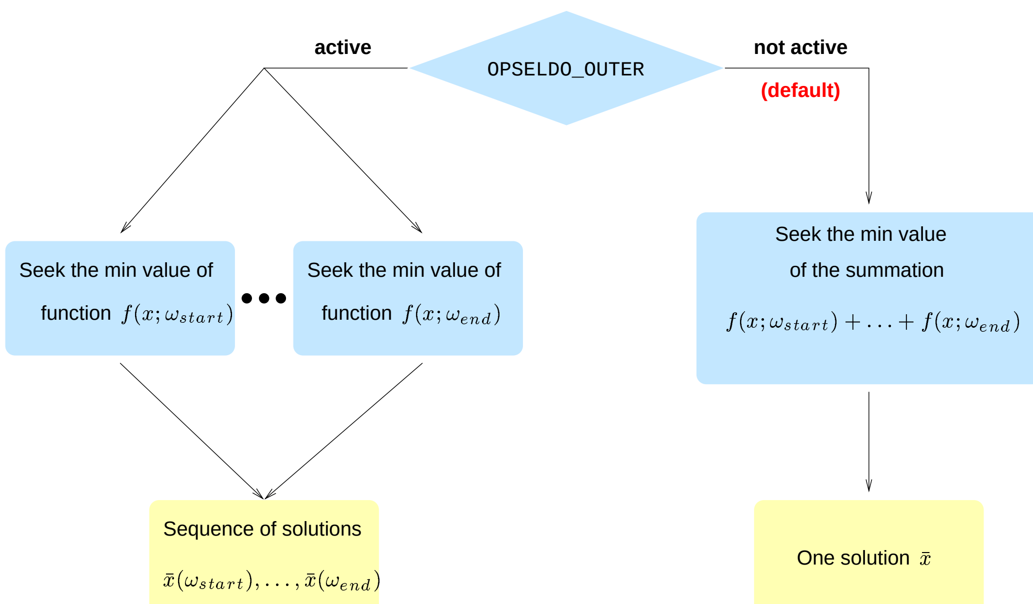
At the same hierarchical level, the design parameters have an order of precedence defined by the simulator. For example, when the netlist contains the following statements, the parameter α will be the "outer-most" and β the "inner-most":

```
.STEP PARAM ALPHA <INCR_SPEC_ALPHA>
.STEP PARAM BETA <INCR_SPEC_BETA>
```

Outer and inner design parameters

The **OPSELDO_OUTER** option allows a reverse behavior of optimization and sweep simulations. It is possible to specify whether the complete set of design parameters defined through a **.STEP PARAM** is 'outer' or 'inner'.

Figure 18-8. Effect of the OPSELDO_OUTER option



Consider the following minimization problem with respect to the *x* variable, where the design parameter ω belongs to the interval $\Omega = [\omega_{start},\omega_{end}]$. A formal circuit formulation could be:

```
.PARAMOPT X=(XINIT, XL, XU)

.STEP PARAM OMEGA OMEAG_START OMEGA_END OMEGA_INCR

.OBJECTIVE
+ EXTRACT_INFO LABEL=f
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE

.OPTIMIZE
```

The strings `XINIT`, `XL`, and `XU` represent, the initial value of the design variable x, its lower bound (or the minimum value that x can take) and respectively its upper bound. Please refer to ".PARAMOPT" on page 1142 for more information.

The previous optimization statements can be understood in two different ways:

- By default, the design parameter ω is always handled as 'inner'. It means that the optimizer will seek the smallest value of the function:

  $$\sum_{\omega_i \in \hat{\Omega}} f(x, \omega_i)$$

  such that the value of x is within the interval $[x_1, x_u]$. This case is represented in the right branch of Figure 18-8.

- When the `OPSELDO_OUTER` option is *active*, the sweep on the design parameter ω becomes external with respect to the optimization process. A sequence of optimizations is launched, one for each of the discretized values within the interval $\Omega = [\omega_{start},\omega_{end}]$.

Note

When the **`OPSELDO_OUTER`** is active, an 'outer' sweep is performed on the complete set of the parameters involved in the **`.STEP`**, **`.DATA`** and **`.TEMP`** commands. The 'outer' and 'inner' design parameters can not be specified at the same time.

Choice of inner and outer parameters

It is possible to specify whether a design parameter defined through a **`.STEP PARAM`** is outer or inner.

The command syntax is a simple extension of the **`.OPTIMIZE`** command:

```
.OPTIMIZE
+ [QUALIFIER=VALUE {, QUALIFIER=VALUE}]
+ [PARAM=LIST_OF_VARIABLES|*]
+ [RESULTS=LIST_OF_MEASURES|*]
```

```
+ [OUTER=LIST_OF_PARAMETERS|*]
```

There are some rules associated to this argument:

- By default, the parameters specified with a **.PARAM** command are inner.
- The order specified within the list of parameters LIST_OF_PARAMETERS has no precedence with respect to the implicit order specified in the netlist.
- For backward compatibility, when the **OPSELDO_OUTER** is active, all the parameters are outer. The list of parameters following the argument **OUTER** are not considered.
- This feature is not valid for the **.STEP** commands using variants of a library (corners), temperature sweeps.

  For example:

  ```
  .OPTION OPSELDO_OUTER
  .STEP PARAM P1 <INCR_SPEC>
  .STEP PARAM P2 <INCR_SPEC>
  ```

  Both parameters are outer parameters:

  ```
  .STEP PARAM P1 <INCR_SPEC>
  .STEP PARAM P2 <INCR_SPEC>
  .OPTIMIZE OUTER=P2
  ```

The parameter P2 is the only candidate for the outer loop. If the arguments of the **.OPTIMIZE** command were as follows **.OPTIMIZE OUTER**=P2,P1 then the parameter P1 would still be the ‘outer-most’ within the outer loop, since this order is specified by the order of **.STEP PARAM** commands.

Specification of multi-points simulations and optimization

The commands that designers can use in order to specify objectives based on multi-point simulations are described in this section. These simulations are the small-signal analysis (**.AC**) and the transient analysis (**.TRAN**). The following specifications do not apply for DC-based design objectives.

Some restrictions are introduced to maintain a reasonable connection between the statements of simulation (**.AC** and **.TRAN** commands) and the optimization objectives (**.OBJECTIVE** command). These limitations are based on two different categories of multi-point analyses:

- **.TRAN** analysis can have *implicit* specification of simulation points, for example:

  ```
  .TRAN TPRINT TSTOP [TSTART [HMAX]]
  ```

  The simulation points are not defined explicitly, since Eldo computes simulation points for accuracy purposes.

- **.TRAN** and **.AC** can have *explicit* specification of simulation points, for example: **.TRAN DATA**=DATA_NAME

The explicit case is certainly the most favorable one since the simulation and the optimization commands can refer to the same list of data. Former versions of the Eldo optimizer used to interpolate measures when the simulation points were implicit. Since this technique introduces additional errors that cannot be bounded, and gives non smooth (or *noisy*) measurements, the default behavior is: when optimization is required, the simulation points (frequency and time) must be included in the set of simulation points.

These limitations are specifically related to the algorithm implemented in the Eldo optimizer/SQP, which cannot cope well with noisy function values (as many algorithms based on Newton iterations). For more information please refer to the section "Smooth and non-smooth problems" on page 1175.

Rules for implicit specification of simulation points

The implicit specification of simulation points appear in transient analyses that use the following statements:

```
.TRAN TPRINT TSTOP [TSTART [HMAX]] [SWEEP DATA=data_name]
.TRAN TPRINT TSTOP [TSTART [HMAX]]
+ [SWEEP PARAM_NAME TYPE NB START STOP]
.TRAN TPRINT TSTOP [TSTART [HMAX]]
+ [SWEEP PARAM_NAME START STOP INCR]
```

It is not possible to specify design objectives that refer to such a category of analyses.

Rules for explicit specification of simulation points

The parameter driven and list **.AC** analysis, and **.TRAN** analysis in `-compat` mode is considered first. These situations match the following statements:

```
.AC TYPE NB FSTART FSTOP [SWEEP DATA=DATA_NAME]
.AC TYPE NB FSTART FSTOP [SWEEP PARAM_NAME TYPE NB START STOP]
.AC TYPE NB FSTART FSTOP [SWEEP PARAM_NAME START STOP INCR]

.AC LIST {LIST_OF_FREQUENCIES} [SWEEP DATA=DATA_NAME]
.AC LIST {LIST_OF_FREQUENCIES} [SWEEP PARAM_NAME TYPE NB START STOP]
.AC LIST {LIST_OF_FREQUENCIES} [SWEEP PARAM_NAME START STOP INCR]

.TRAN INCR1 T1 [{INCRN TN}] [TSTART=VAL] [SWEEP DATA=DATA_NAME]
.TRAN INCR1 T1 [{INCRN TN}] [SWEEP PARAM_NAME TYPE NB START STOP]
.TRAN INCR1 T1 [{INCRN TN}] [SWEEP PARAM_NAME START STOP INCR]
```

An objective statement will satisfy the command described in section ".OBJECTIVE" on page 1134. The arguments **GOAL**, **EQUAL**, **LBOUND**, **UBOUND**, **WEIGHT**, and **TYPVAL** must be a scalar value or a constant waveform.

When the waveforms are used in `OBJECTIVE_INFO`, the value of the objectives (goal and bounds) are sampled values of the waveform at simulation points, and it is possible to mix scalar and waveforms. Specifying `LBOUND` as a scalar means that this bound is global.

Now consider the data-driven **.AC** and **.TRAN** analyses:

```
.AC DATA=data_name [SWEEP DATA=data_name]
.AC DATA=data_name [SWEEP param_name TYPE nb start stop]
.AC DATA=data_name [SWEEP param_name start stop incr]

.TRAN DATA=data_name [SWEEP DATA=data_name]
.TRAN DATA=data_name [SWEEP param_name TYPE nb start stop]
.TRAN DATA=data_name [SWEEP param_name start stop incr]
```

An objective statement of **GOAL** or **EQUAL** should satisfy the following syntax:

```
.<ANALYSIS> DATA=OMEGA_DATA(OMEGA)

.DATA OMEGA_DATA OMEGA GOAL_VAl WEIGHT_VAL TYP_VAL
+ OMEAG1 GOAL_VAL1 WEIGHT_VAL1 TYP_VAL1
+ OMEGA2 GOAL_VAL2 WEIGHT_VAL2 TYP_VAL2
... ... ... ...
+ OMEGAP GOAL_VAL1P WEIGHT_VALP TYP_VALP
.ENDDATA

.OBJECTIVE
+ EXTRACT_INFO [LABEL=NAME]
+ {$MACRO|FUNCTION}
+ OBJECTIVE_INFO
+ [SCALE_INFO]
```

where the parameter `OMEGA` is the keyword **TIME** or **FREQ**, and

```
SCALE_INFO:=
TYPVAL=RVALUE
```

and

```
OBJECTIVE_INFO:=
{GOAL=OMEGA_DATA(GOAL_VAL) | RVALUE}
| {EQUAL=OMEGA_DATA(GOAL_VAL) | RVALUE}
[ WEIGHT=OMEGA_DATA(WEIGHT) | RVALUE]
[ TYPVAL=OMEGA_DATA(TYP_VAL) | RVALUE]
```

Using the same **.DATA** statement in the analysis and the objective statement ensures coherency of data. It is possible to specify a scalar value (denoted by `RVALUE`) for the **GOAL**, **EQUAL**, **WEIGHT**, and **TYPVAL** arguments.

Note

When **WEIGHT** and **TYPVAL** are scalar values, they apply to all lines.

An objective statement with the arguments **LBOUND** or **UBOUND** type will satisfy the following syntax:

```
.<ANALYSIS> DATA=OMEGA_DATA

.DATA OMEGA_DATA OMEGA L_VAL U_VAL WEIGHT_VAL TYP_VAL
```

```
+ OMEGA1 L_VAL1 U_VAL1 WEIGHT_VAL1 TYP_VAL1
+ OMEGA2 L_VAL2 U_VAL2 WEIGHT_VAL2 TYP_VAL2
...
+ OMEGAP L_VALP U_VALP WEIGHT_VALP TYP_VALP
.ENDDATA
```

with:

```
OBJECTIVE_INFO:=
{{LBOUND=OMEGA_DATA(L_VAL)|RVALUE} | {UBOUND=OMEGA_DATA(U_VAL)|RVALUE}}
| {LBOUND=OMEGA_DATA(L_VAL)|RVALUE} {UBOUND=OMEGA_DATA(U_VAL) | RVALUE}
[WEIGHT=OMEGA_DATA(WEIGHT_VAL) | RVALUE]
```

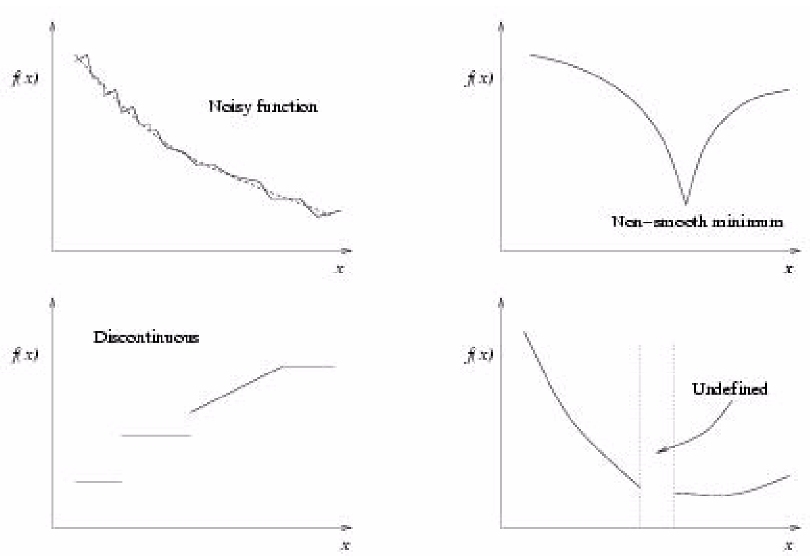
and

```
SCALE_INFO:=
TYPVAL={OMEGA_DATA(TYP_VAL) | RVALUE}
```

Smooth and non-smooth problems

A difficult situation arises when the functions corresponding to the design objectives do not have continuous derivatives or are not continuous at all. In this case, methods for smooth problems (such as the Eldo optimizer/SQP algorithm) will encounter difficulties.

Figure 18-9. Examples of non smooth problems



It is assumed in the optimization problem, that the functions are continuous with continuous derivatives on some domain (or sufficiently smooth). Figure 18-9 illustrates some of these difficulties:

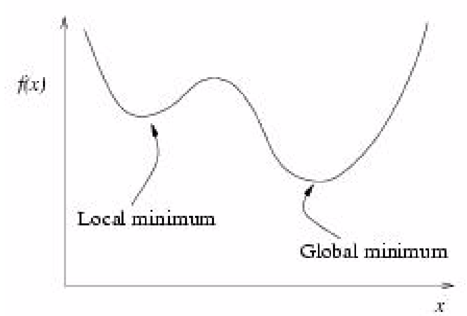
- The first case represents a “noisy function” with an overall “trend” plotted with dashed lines. It is related to the effect of features such as adaptive algorithms and stopping tests in iterative methods inside the simulation. Note that the “noise” is not stochastic in such situations. The Eldo optimizer/SQP will fail to make any progress because local descent directions may point uphill.

- The second example is related to the use of functions such as ABS(.), MIN(.) or MAX(.) that are not differentiable in the common sense. These situations can represent minor difficulties when the user wants only improvement of the current solution rather than optimization at “full-blown optimality”.
- In general, the last two cases will lead to serious difficulties. The “discontinuous” case probably involves functions that are not numerical in nature, or the discontinuities are the result of features such as table look-ups and switches. On the other hand, the “undefined” case may be the result of unexpected failures of the simulation or undefined extracted measures.

Global and Local Optimization

The fastest optimization algorithms only search for a *local* solution at a point at which the objective function is smaller than all other feasible points in its vicinity. They don’t always find the best of all such minima, that is the *global* solution (see Figure 18-10). The reason for this is that practical methods for finding global optima are (at present time) too expensive in all but the most specialized cases.

Figure 18-10. Different types of minima



General non-linear problems may possess local solutions that are not global solutions. Global solutions are highly desirable, however, they are usually difficult to identify and even more difficult to locate. Unless very strong assumptions are made about the functions that define the optimization problem in question, characterizations of global minima are almost impossible, so even if one is found the user may never know.

The algorithm implemented in the Eldo optimizer/SQP is based on Newton iterations. It is therefore, not able to combine local and global searches, and can only find local solutions.

Discrete optimization

In some optimization problems, the variables only make sense if they take on integer or discrete real values. The obvious strategy of ignoring the integrality requirement, solving the problem with real variables. Rounding all the components to the nearest integer is not guaranteed to give a solution that is close to optimal. Problems of this type should be handled using the tools of discrete optimization. The mathematical formulation is changed by adding the constraint "x_i integer for all $i \in K$", where K is a subset of {1,...,N}.

Continuous optimization problems are easier to solve, because the smoothness of the functions makes it possible to use the *objective* function and constraint information at a particular point x to deduce information of the functions behavior at all points close to x. The same statement cannot be made about discrete problems. Where points that are close in some sense may have markedly different function values. Moreover, the set of possible solutions is too large to make an exhaustive search for the best value in this finite set. When models contain variables allowed to vary continuously and others that can attain only integer values, there are referred to as mixed-integer programming problems (MIP).

The actual Eldo optimizer can only treat Nonlinear Problems (NLP) with continuous variables. The increment argument in the **`PARAMOPT`** command is used to define a grid of "integer feasible" points with uniform steps. It should be noted that "the obvious strategy of ignoring the requirement of discretization is used, solving the problem with real variables, and then rounding all the components to the nearest point onto the grid at the end of optimization." This strategy is based on the assumption that the increment value is sufficiently small. The problem solved by the optimizer is a "continuous relaxation| of the original MIP problem. The constraints of integrality on variables are simply relaxed.

Scaling of variables and objectives

An important consideration in solving many circuit design problems is that some extracted measures or optimization variables may vary greatly in magnitude.

Scaling of variables

For example, the user may have a minimization problem in which the first independent variable $x^{(1)}$ is in the range $[10^8, 10^9]$ Hz and the second $x^{(2)}$ in the range $[10^{-7}, 10^{-6}]$ seconds. These ranges are referred to as the *scales* of the respective variables. In this section, the effect of such widely disparate scales on the algorithms is considered.

One place where scaling will effect our algorithms is in calculating the difference between iterates. In the above example, any such calculation will virtually ignore the second variable (the time). The obvious strategy of re-scaling the optimization variables is used in the Eldo optimizer/SQP; that is, change their units. For example, if the units of $x^{(1)}$ are changed to Giga Hertz and the units of $x^{(2)}$ are changed to microseconds, then both variables will have the range [0.1, 1] and the scaling problem in computing the differences will be eliminated. Notice that this corresponds to changing the independent variable to

$$\hat{x} = D^{-1} \cdot x$$

where D is a diagonal matrix of scaling.

The Eldo optimizer/SQP method uses an affine scaling transformation based on the values of the upper and lower bounds. If the variable x satisfies the relations 1 ≤ x ≤ u, one can introduce the two numbers:

$$D = \frac{2}{u-l}, c = \frac{u+l}{u-l}$$

that define the following transformation:

$$\hat{x} = D \cdot x - c$$

With this formula the transformed variable will have range [-1, 1]. The user must set the appropriate bounds on the variable x corresponds to the desired change in units, and then the algorithms operate as if they were working in the transformed variable space.

The result of this scaling procedure is described in the major output file *.otm*. The section in the *.otm* file begins with the heading “`Section 2 - Scaling Transformation For Variables`”. Please refer to “ASCII Output for Eldo Optimizer/SQP” on page 1191 for more information.

The above scaling strategy is not always sufficient; for example, there are cases that need more appropriate methodology. Consider the following problem:

```
* Circuit Statements
.PARAM LOGL={LOG10(150U)}
.PARAM LOGR={LOG10(10)}
.PARAM LOGC={LOG10(10U)}

V1 1 0 AC 1
L1 1 2 5U
L2 2 3 150U
C1 3 0 33U

LA 2  2A {10^LOGL}
CA 2A 2B {10^LOGC}
RA 2B 3  {10^LOGR}

.AC DEC 500 1E2 1E5

* Optimization Commands
.OPTIMIZE

.PARAM LNOM={LOG10(150U)} LMIN={LOG10(150U)} LMAX={LOG10(1)}
.PARAMOPT LOGL=(LNOM,LMIN,LMAX)

.PARAM CNOM={LOG10(10U)} CMIN={LOG10(1P)} CMAX={LOG10(1)}
.PARAMOPT LOGC=(CNOM,CMIN,CMAX)

.PARAM RNOM={LOG10(10)} RMIN={LOG10(1E-3)} RMAX={LOG10(1E+6)}
.PARAMOPT LOGR=(RNOM,RMIN,RMAX)

.OBJECTIVE AC LABEL=Q MAX(W('VM(3)'))
+ GOAL=MINIMIZE
```

This example represents a user-defined transformation based on the logarithmic function, such that:

$$x = 10^{\hat{x}}$$

The exponent becomes the parameter to optimize. Consider the third **.PARAMOPT** command (on the previous page). The original formulation would be the following statements:

```
.PARAM RNOM=10 RMIN=1E-3 RMAX=1E+6
.PARAMOPT R=(RNOM, RMIN, RMAX)
```

In this case, the affine scaling transformation is not appropriate, because the upper bound RMAX=10^6 will dominate inside the expression of the matrix D:

$$D = \frac{2}{RMAX - RMIN} \approx \frac{2}{RMAX}$$

Scaling of design objectives

It is also necessary to consider the scale of the design objectives. The scale of the design objectives matter in the stopping conditions and the computation of derivatives. Differing sizes among the component functions of an extracted measure F can cause the same types of problems as differing sizes among the optimization variables.

For instance, the algorithm requires a decrease in some merit function, and it is clear that if the units of two component functions of F(x) are widely different, then the smaller component function will be virtually ignored. For this reason, our algorithms also use a positive diagonal scaling matrix Df on the design objectives F(x), which works as D does on x. The diagonal matrix Df is chosen so that all the components of Df F(x) will have about the same typical magnitude. In our interface, the user specifies Df initially by using the argument **TYPVAL**. The algorithm then sets:

$$Df = \frac{1}{TYPVAL}$$

The result of this scaling procedure is described in the major output file *.otm*. The section in the *.otm* file begins with the heading `Section 1- Scaling Transformation Design Objectives`. Please refer to “ASCII Output for Eldo Optimizer/SQP” on page 1191 for more information.

Minimization and Maximization of Objectives

In order to simplify the presentation the case **GOAL=MINIMIZE** will be considered. The Eldo optimizer always minimizes, thus “maximizing f(x)” is handled as “minimizing –f(x)”.

Command syntax

The functions (or extracted measures) that the user wants to minimize have to be specified by complementing the **.OBJECTIVE** command as follows:

```
* Minimize or Maximize statements
.OBJECTIVE EXTRACT_INFO LABEL=f_m
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE
+ [WEIGHT=MU_M]
```

The optional weight μ_m is a positive number. By default, this number is initialized to one.

Multiple specifications

This syntax allows multiple specifications of *objective* functions in the netlist. If several objectives are qualified as **GOAL**=**MINIMIZE** (or **MAXIMIZE**), a *weighted* objective function will be built as the summation of all these particular measures. It means that the optimizer will minimize a unique objective that is the weighted sum of the whole objectives. Consider the following:

```
* Aggregation of two Minimize statements
.OBJECTIVE EXTRACT_INFO_1 LABEL=F_M_1
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE WEIGHT=MU_M_1

.OBJECTIVE EXTRACT_INFO_2 LABEL=f_M_2
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE WEIGHT=MU_M_2
```

These statements, where two design objectives have to be minimized, define a *bi-criterion* problem or a *vector-valued* objective function ($f_m^{(1)}$, $f_m^{(2)}$). The technique used in this circumstance is a standard approach for finding solution of a vector optimization problem. This technique is named *scalarization*. Applied on our simple problem, it leads to the minimization of the global objective function:

$$F = \mu_m^{(1)} f_m^{(1)} + \mu_m^{(2)} f_m^{(2)}$$

In other words, the optimization method considers a *unique* problem that is the aggregation of two concurrent problems.

In practice users have to experiment with the different choices of weights by successive adjustments. This technique is explained in "Role of the weight numbers" on page 1181.

Effect of multiple sweeps and step increments

When multiple sweeps or multiple step increments on circuit parameters is present in the netlist, an extracted measure qualified as **GOAL**=**MINIMIZE** (or **MAXIMIZE**) must be considered as a multidimensional object. Consider the following statements where a number of P parameters have been specified:

```
* Design parameters specification
.STEP PARAM OMEGA_J <INCR_SPEC_OMEGA_J> ! for all j ∈ {1, …, P}

* Minimize or Maximize statements
.OBJECTIVE EXTRACT_INFO LABEL=f_m
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE
+ WEIGHT=MU_M
```

As above, the technique of *scalarization* is used, replacing the minimization of the vector-valued function by the minimization of the sum of the components of f_m. The optimizer will then form and minimize the function:

$$F_m(x) = \mu_m \sum_{j(1), \ldots, j(P)} f_m(x;\omega^{j(1)}, \ldots, \omega^{j(P)})$$

The extracted measure f_m can be considered as a *group of functions* that has to be minimized.

Note

With this kind of construct the weight number applies to all functions in the group. In other words, these functions have the same relative importance.

Role of the weight numbers

The numbers $\mu_m^{(i)}$ are *positive* weight values attached to each design objective. For instance, the weight $\mu_m^{(i)}$ can be thought of as quantifying the user's desire to make $f_m^{(i)}(x)$ small. The user would take $\mu_m^{(i)}$ large if the user wants $f_m^{(i)}(x)$ to be small. The ratio $\mu_m^{(i)} / \mu_m^{(j)}$ can be interpreted as the relative weight of the *i*th objective compared to the *j*th objective.

These remarks can help users that want to change the weights in order to get the lower values of a chosen objective, for example the *k*th. To find an optimal point which trades off the *k*th objective, users can increase the weight on the *k*th objective.

Goal Values for Objectives

A design objective that represents an equality specification for a performance measurement f_r such that the extracted value of f_r is as close as possible to a goal value r. This design objective can be defined with **`GOAL`** argument.

Command syntax

The user can specify goal values for extracted measures by complementing the **`.OBJECTIVE`** command as follows:

```
* Goal values on measures
.OBJECTIVE EXTRACT_INFO LABEL=F_R
+ {$MACRO|FUNCTION}
+ GOAL=R
```

```
+ [WEIGHT=MU_R]
```

The optional weight μ_r is a positive number. By default, this number is initialized to one. These functions are handled using the *non-linear least squares* approach, that is by minimizing the squared difference between the actual value of the measure f_r and the goal value r: $\mu_r(f_r - r)^2$

Multiple specifications

As previously stated, the extracted measures specified with **GOAL**=R are handled as the **GOAL**=**MINIMIZE** objectives, the following can be considered:

```
* Aggregation of two Minimize statements
.OBJECTIVE EXTRACT_INFO_1 LABEL=F_R_1
+ {$MACRO|FUNCTION}
+ GOAL=R_1 WEIGHT=MU_R_1

.OBJECTIVE EXTRACT_INFO_2 LABEL=F_R_2
+ {$MACRO|FUNCTION}
+ GOAL=R_2 WEIGHT=MU_R_2
```

This leads to the minimization of the global objective function:

$$F = \mu_r^{(1)} (f_r^{(1)} - r^{(1)})^2 + \mu_r^{(2)} (f_r^{(2)} - r^{(2)})^2$$

In practice users have to experiment with different choices of weights by successive adjustments (please refer to “Role of the weight numbers” on page 1181).

Effect of multiple sweeps and step increments

The combination of optimization with goal values and multiple **.STEP** commands is supported. Consider the following statements where P parameters have been specified:

```
* Design parameters specification
.STEP PARAM OMEGA_1 <INCR_SPEC_OMEGA_1>
.STEP PARAM OMEGA_2 <INCR_SPEC_OMEGA_2>
...
.STEP PARAM OMEGA_P <INCR_SPEC_OMEGA_P>

* Goal statement
.OBJECTIVE EXTRACT_INFO LABEL=F_R
+ {$MACRO|FUNCTION}
+ GOAL=R
+ WEIGHT=MU_R
```

As above, the technique of *scalarization* is used, replacing the minimization of a vector-valued function by the minimization of the sum of the components of f_r. The optimizer will then form and minimize the function:

$$F_r(x) = \mu_r \sum_{j(1), \ldots, j(P)} (f_r(x; \omega^{j(1)}, \ldots, \omega^{j(P)}) - r)^2$$

Note

The weight number applies to all functions in the group, however, it is possible to associate a weight to each component of f_r. This can done with the **.DATA** command. Please refer to "Rules for explicit specification of simulation points" on page 1173 for more information.

Range Constraints on Objectives

It is possible to specify bounds on design objectives. These objectives are named inequality constraints or range constraints. In this case the lower and upper bounds are equal l = u.

Command syntax

Inequalities on measures correspond to relations like $l \le f_i(x) \le u$ these constraints are specified by complementing the **.OBJECTIVE** command as follows:

```
* Range constraint
.OBJECTIVE EXTRACT_INFO LABEL=F_I
+ {$MACRO|FUNCTION}
+ LBOUND=L UBOUND=U
```

When the objective f_i does not need to be bounded by a lower or an upper value, the arguments may be omitted. For example, if a design objective must be positive $f_i \ge 0$ (its lower bound is 0), the user can specify the following:

```
* Positive objective
.OBJECTIVE EXTRACT_INFO LABEL=F_I
+ {$MACRO|FUNCTION}
+ LBOUND=0.0
```

The Eldo optimizer/SQP algorithm handles them directly as "hard constraints". These objectives must be satisfied at the solution of the problem.

Notes

Before examining the effect of multiple sweep commands on constraints, some notes are necessary:

- It is important to quote: the iterates (the successive values of design parameters taken during optimization) are by no means guaranteed to be feasible with respect to these constraints. If the optimization problem appears to be (hopefully) feasible, these will be satisfied at the solution, or in the vicinity of a solution.
- In general, the inequality constraints are more relevant than the equality constraints, because equalities may be difficult to satisfy. For instance, it is not possible to specify a tolerance associated to equality constraints. Consider the objective of fixing the Input Power Matching at a frequency of 2.4GHz of a Low Noise Amplifier:

```
.EXTRACT AC LABEL=S11_dB@2.4GHz YVAL(SDB(1,1),2.4G)
+ GOAL=-15
```

It is obvious that a more appropriate objective would be to specify a target value with some additional tolerance, for example 1dB. The actual implementation of Eldo optimizer/SQP does not permit this feature. As a workaround, the user can combine **GOAL** objectives in order to center the design and range constraints with the same objective but on separate statements.

- As a consequence of the previous limitation, the equalities and inequalities are mutually exclusive. This means that the argument `EQUAL` cannot appear inside the same objective statement with the arguments `LBOUND` and `UBOUND`.
- Another type of condition that is not included in problem (P), is a constraint of the form $f > 0$. It is difficult to handle constraints of this form in an active set method such as the Eldo optimizer/SQP algorithm, because the feasible region is not closed and the constraint cannot be active at a solution. However, it can be useful to include them in the problem via the transformation $f(x) \geq \varepsilon > 0$, possibly solving a sequence of problems in which $\varepsilon \rightarrow 0$ if the constraints happen to be active. The reason for doing this may be to prevent or dissuade f(x) being evaluated at an infeasible point at which it is not defined.

Multiple specifications

Multiple range constraints can be specified. For example, if the netlist consists of two statements such as:

```
* Aggregation of two range constraints
.OBJECTIVE EXTRACT_INFO_1 LABEL=F_I_1
+ {$MACRO|FUNCTION}
+ LBOUND=L_1 UBOUND=U_1

.OBJECTIVE EXTRACT_INFO_2 LABEL=F_I_2
+ {$MACRO|FUNCTION}
+ LBOUND=L_2 UBOUND=U_2
```

These objectives will be optimized jointly, meaning that a vector of design variables x has to satisfy the system of inequalities:

$$\begin{pmatrix} l^{(1)} \\ l^{(2)} \end{pmatrix} \leq \begin{pmatrix} f^{(1)}(x) \\ f^{(2)}(x) \end{pmatrix} \leq \begin{pmatrix} u^{(1)} \\ u^{(2)} \end{pmatrix}$$

This approach consists of defining a unique optimization problem that is the aggregation of all the constraint statements.

Effect of multiple sweeps and step increments

As above, the combination of optimization with range constraints and multiple **.STEP** commands can be specified. Consider the following statements where P parameters have been specified:

```
* Design parameters specification
.STEP PARAM OMEGA_1 <INCR_SPEC_OMEGA_1>
.STEP PARAM OMEGA_2 <INCR_SPEC_OMEGA_2>
...
.STEP PARAM OMEGA_P <INCR_SPEC_OMEGA_P>

* Range cosntraint statement
.OBJECTIVE EXTRACT_INFO LABEL=f_i
+ {$MACRO|FUNCTION}
+ LBOUND=L
+ UBOUND=U
```

The optimizer will then form a group (or system) of inequalities as shown below:

$$\forall j(1), \ldots, j(P), \begin{cases} l(\omega^{j(1)}, \ldots, \omega^{j(P)}) \le f_i(x;\omega^{j(1)}, \ldots, \omega^{j(P)}) \\ f_i(x;\omega^{j(1)}, \ldots, \omega^{j(P)}) \le u(\omega^{j(1)}, \ldots, \omega^{j(P)}) \end{cases}$$

The Optimization Methods

This section is concerned with the optimization command acting on the Eldo optimizer/SQP and the Eldo optimizer/Search algorithm parameters. In some specific cases, designers can perform optimization using the Eldo optimizer/*Dichotomy*, *Secant* and *Passfail* methods. Please refer to "Eldo Optimizer/Dichotomy/Secant/PassFail Arguments" on page 1140 to set-up a measurement with Eldo using this method.

Eldo Optimizer/SQP Method

The Eldo optimizer/SQP optimizer is based on a *Sequential Quadratic Programming* method (SQP), using an active-set algorithm (proposed by Fletcher [2]) for solving large-scale quadratic programming problems. It is efficient for solving a large class of smooth optimization problems belonging to the following list:

- unconstrained and bound constrained problems (using only the statements **GOAL**=**MINIMIZE**|**MAXIMIZE** and **GOAL**=RVALUE)
- systems of non-linear equations (using only the statement **EQUAL**=RVALUE)
- general (equality and inequality) constrained problems.

Users who wish to consult recent books giving a broader view of optimization techniques should refer to Nocedal and Wright [5], and Fletcher [3]. For more advanced books one may consult Bonnans, Gilbert, Lemarechal and Sagastizabal [1] and Gill, Murray and Wright [4].

Before continuing our presentation, some useful definition and mathematical notations are given. The design variables are denoted by:

$x = (x^{(1)}, x^{(2)}, ..., x^{(N)})$

a vector of real numbers of dimension *N*. Therefore the space of variables is denoted by $\Re^N$. The scalar product is denoted by:

$\langle u|v \rangle = \sum_i u^{(i)} v^{(i)}$ onto the space $\Re^N$, and its associated norm is $\|u\|$.

A simplified structure of the *k*th iteration can be depicted as:

- Step Computation: determine a direction of search s_k (by solving a tangent quadratic subproblem $(QP)_k$,)
- Step Assessment: find $\alpha_k > 0$ in order to minimize a chosen merit function $x \rightarrow \Phi(x)$ along the line $\alpha \rightarrow x_k + \alpha s_k$,
- Update: $x_{k+1} = x_k + \alpha_k s_k$.

This approach is referred to as a *descent method* since the search direction s_k satisfies a descent property:

$\langle s_k | \nabla\Phi_k \rangle < 0$

where the vector $\nabla\Phi_k$ represents the gradient of the chosen merit function. The role of the line search algorithm is to *dampen* the displacement s_k. Note that during the optimization process simulations are required at two distinct stages:

- the computation of *derivatives* of the extracted measures involved in the problem statement,
- the step assessment procedure, or *line search* algorithm

Role of tolerances in Eldo optimizer/SQP

This section gives some indications on the role of the arguments **TOL_GRAD**, **TOL_FEAS**, and **TOL_OPT**.

Suppose a value of x that is a local minimizer of f(x) in the interval $a \le x \le b$ is to be obtained:

Minimize	f(x)
Subject to	$a \le x \le b$

or using a SPICE formulation:

```
.OPTIMIZE
* Minimize Statement
```

```
.OBJECTIVE EXTRACT_INFO LABEL=F
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE

* Design variable specification
.PARAMOPT X=(X0, A, B)
```

The optimality conditions OPTIM(x) in the definition of the **TOL_GRAD** argument are:

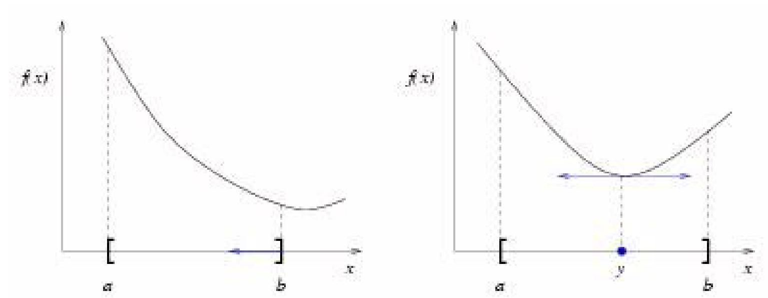
	f'(x) if a < x < b
OPTIM(x)	min(f'(x),0) if x = a
	max(f'(x),0) if x = b

Consider the first case in Figure 18-11. The minimum is the point `x = b`. The blue vector represents the derivative f'(x) at point b (this is the opposite of the steepest descent direction -f'(x) at point b). The derivative f'(x) is negative, then is the number max(f'(x),0)=0. When OPTIM(x) is zero or very small this indicates that the point x is an optimum. The number **TOL_GRAD** is used in the stopping test of the Eldo optimizer/SQP. The point x is optimal when the absolute value of OPTIM(x) is less than **TOL_GRAD**.

Consider the second case in Figure 18-11. Point y is an unconstrained minimizer of f, since it lies strictly in the interval [a,b]. The optimality condition is then OPTIM(y)=f'(y) which is the slope of f at point y. OPTIM(y) equals zero. Note that the previous definition of the optimality conditions are related only to bound constrained minimization problems. The conditions OPTIM(x) are extended to the more general problems that the Eldo optimizer/SQP can treat.

In many cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost.

Figure 18-11. Illustration of the optimality conditions



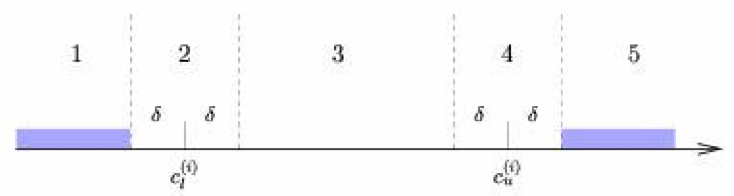
For instance, suppose that a constraint function $c^{(i)}(x)$ is computed for some relevant *x* and if the first six digits are known to be correct. A constraint should be considered as active at its upper bound (or lower bound), if the magnitude of the difference between the values $c^{(i)}(x)$ and $c_u^{(i)}$ (or $c_l^{(i)}$ respectively) is less than some tolerance of order 1.0×10^{-6}.

This tolerance, δ, specifies how accurately the constraints should be satisfied. It defines the maximum absolute violation in non-linear constraints at a feasible point.

> A constraint is considered satisfied if its violation does not exceed the tolerance δ.

The feasible region for the constraints $c_l^{(i)} \leq c_I^{(i)}(x) \leq c_u^{(i)}$ is shown in Figure 18-12.

Figure 18-12. Illustration of the constraints $c_l^{(i)} \leq c^{(i)}(x) \leq c_u^{(i)}$



The constraints are considered satisfied if $c^{(i)}(x)$ lies in the region 2, 3, or 4, and inactive if $c^{(i)}(x)$ lies in region 3. The constraint $c_l^{(i)} \leq c^{(i)}(x)$ is considered active in region 2, and violated in region 1. Similarly, $c^{(i)}(x) \leq c_u^{(i)}$ is active in region 4, and violated in region 5. For equality constraints $c_l^{(i)} = c_u^{(i)}$, regions 2 and 4 are the same, and region 3 is empty. The default value is appropriate when the constraints contain data about the accuracy.

Note that specifying an appropriate tolerance on feasibility **`TOL_FEAS`** may lead to several savings, by allowing the optimization procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

Computation of finite-difference derivatives

One of the most demanding phase in terms of simulation is the computation of derivatives. The finite differences in the Eldo optimizer are used. Finite differencing is an approach to calculate the approximate derivatives whose motivation comes from Taylor's theorem. Like many software packages, the Eldo optimizer performs automatic calculation of finite differences whenever the simulator is unable (or unwilling) to supply the code to compute exact derivatives.

A popular formula for approximating the partial derivative $\partial F/\partial x$ at a given point is the *forward differences* or *one-sided differences.* The parameter denoted by h_f is used, this controls the interval used to estimate the gradients of the function F by forward differences:

$$\frac{\partial F}{\partial x} \approx \frac{F(x + h_f) - F(x)}{h_f}$$

One-sided difference estimates are used to ensure feasibility with respect to an upper or lower bound on *x*. If *x* is close to an upper bound, the trial intervals will be negative. The final interval is always positive.

- An approximation to the derivative of F can be obtained by evaluating the function F at N + 1 points and performing some elementary arithmetic.
- The resulting gradient estimates should be accurate to $O(h_f)$ unless the functions are badly scaled.

Eldo Optimizer/Search Method

The AMS 2006.1 release contains a new method of optimization that complete the previous ones. This method is named *Search* and is a combination of bisection and inverse quadratic interpolation. It is specified with the command **`.OPTIMIZE METHOD=SEARCH`**. It allows minimization in the case of one dimensional problems. The *Search* method belongs to the class of derivative free optimization (DFO) algorithms.

The *Search* method has been introduced for completing the Dichotomy and Secant methods. It is also dedicated for solving problems having one variable and one extracted measure problems. This approach will run faster than the Dichotomy method and would be more robust than the modified Dichotomy (the Eldo optimizer/Secant).

The *Search* method is based on two distinct algorithms: `FIND_ZERO` and `FIND_MINIMUM`. The algorithm `FIND_ZERO` aims to find a solution to:

$$\text{Find } x \text{ such that} \quad F(x)=0 \quad \text{where } a \le x \le b$$

or using a SPICE formulation:

```
.OPTIMIZE METHOD=SEARCH

* Design variable specification
.PARAMOPT X=(X0, A, B)

* Goal value
.OBJECTIVE <ANALYSIS> LABEL=F_R <FUNCTION> GOAL=R
```

Note

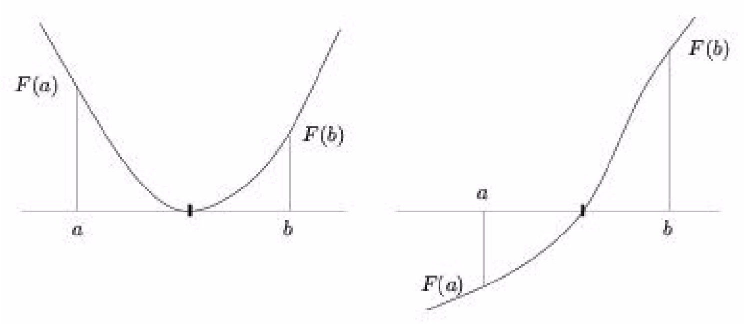
Specifying a weight number with **`WEIGHT`**$=\mu_r$ is allowed, however, it will not be considered. A warning message will be generated in this very specific case.

The most important feature of the method `search` can be stated in the following form:

- when the interval [a,b] is not specified, the guess point x0 is mandatory. One must have the command: `.PARAMOPT x=(x0,*,*)`. The method `search` enters a search phase for finding an interval locating a solution.
- When the interval [a,b] is specified, the method will start directly the search for a solution. The guess point is not used.

$F(x)= f_r(x) - r$ defines the difference between the actual measure f_r and the goal value r. The `FIND_ZERO` algorithm uses a combination of bisection, secant, and inverse quadratic interpolation methods. The `FIND_ZERO` algorithm defines a zero as a point where the function crosses the x-axis. Points where the function touches but does not cross (see Figure 18-13), the x-axis are not valid zeros. For example, $F(x)= x^2$ is a parabola that touches the x-axis at (0,0). Since the function never crosses the x-axis, no zero is found. For functions with no valid zeros, **`FIND_ZERO`** executes until an undefined value is detected.

Figure 18-13. Non-valid and valid zeros



The algorithm `FIND_MINIMUM` attempts to return a value of x that is a local minimizer of F(x) in the interval $a \le x \le b$:

Minimize	F(x)
Subject to	$a \le x \le b$

The algorithm is based on the golden section search and parabolic interpolation techniques.

Exploiting Output Results

After the optimization algorithm has been applied, the designer must be able to recognize whether it has succeeded in its task of finding a solution. This can be accomplished by analyzing the results of the optimization.

Binary Output

```
.PLOT OPT WOPT(label_name) [(LOW, HIGH)] [(VERSUS)]
+ {OVN [(LOW, HIGH)]} [(SCATTERED)]
```

Parameters

- **OPT**

  Optimization analysis. The waves produced by this analysis can only be specified using the keyword `WOPT`.

- `WOPT`

  Only available for extracting waves generated during an optimization process. These are implicitly declared waves which have the same name as the extract label they refer to. They represent extract results versus the index of the run.

It is possible to plot extracted waves generated during an optimization process. These are implicitly declared waves which have the same name as the extract label they refer to. They represent extract results versus index of the run.

Example

```
.EXTRACT tran label=FOO yval(v(node),3n) !Define the extract
.PLOT OPT WOPT(FOO)
```

ASCII Output for Eldo Optimizer/SQP

Starting Eldo v6.7, the ASCII output were defined as follows:

- the standard output controlled by Eldo is dedicated to simplified information on the optimization process.
- the optimization file *.otm* is organized in three major sections: initialization (scaling of variables and objectives), optimization phase, and the results of the process (final variables and objectives).

The following example will be considered. It is a slight modification of the LNA example (consult "Designing a Low Noise Amplifier (LNA)" on page 1149). The modified commands are shown below:

```
* Input Power Matching at Frequencies 2.4GHz and 2.5GHz

.EXTRACT AC LABEL=S11_dB@2.4GHz YVAL(SDB(1,1),2.4G) LBOUND=-15 UBOUND=-10
.EXTRACT AC LABEL=S11_dB@2.5GHz YVAL(SDB(1,1),2.5G) LBOUND=-15 UBOUND=-10
```

where the **GOAL** objective is replaced with two range constraints.

Standard output

The standard output only involves the major phase of the optimization process. For each iteration Eldo will print a line of data. This line gives the cumulative number of simulations, and some algorithmic informations on the progress of optimization:

- `Iter`: major iteration number.
- `Simul`: cumulative number of simulations performed at the current iteration.
- `LS Step`: The step length α_k taken the direction s_k at the k-*th* iteration. This step length is used to update the current iterate with the formula: $x_{k+1} = x_k + \alpha_k s_k$

  When the problem is well defined, this number should tend to 1.0 at the end of optimization. Failure of optimization can be detected with repetitive small steps.
- `Merit Function`: value of the merit function. This number will decrease as the solution is approached.
- `Feas`, `Optim`: these positive numbers are FEAS(x) and OPTIM(x) respectively. Both of these should be small in the vicinity of an optimal point.
- `Slack`: represents the norm of the slack variables added to the range constraints in order to prevent a potential inconsistency.

The following output has been obtained after running Eldo on the LNA example on the previous page:

```
Optimization Phase
 ==================

 ==> Detailed results in the major output file .otm

 ==> Output in the following order :
     Iter    ... Major iteration number
     Simul   ... Cumulative number of simulations
     LS Step ... Steplength parameter in linesearch
     Merit F ... Value of the merit function
     Feas    ... Constraint violation (measure of feasibility)
     Optim   ... Optimality criterion
     Slack   ... Norm of variables to prevent inconsistency


 ==> Starting Optimization...


  Iter Simul    LS Step   Merit Function    Feas    Optim    Slack
     1     8    1.0e+00    9.4580268e+01  3.7e+00           0.0e+00
     2    19    6.3e-02    9.1491101e+01  3.1e+00  2.0e+00  0.0e+00
     3    31    3.5e-01    5.8178533e+01  2.9e+00  2.0e+00  0.0e+00
     4    41    1.7e-01    3.8139395e+01  1.0e+00  2.0e+00  0.0e+00
     5    51    3.4e-01    1.3713556e+01  7.3e-01  2.0e+00  0.0e+00

...
```

```
 Iter Simul    LS Step   Merit Function    Feas     Optim    Slack
   81   797    1.0e+00    4.3858634e+00  0.0e+00  1.4e-03  0.0e+00
   82   808    1.0e-00    4.3858634e+00  0.0e+00  1.4e-04  0.0e+00


==> Optimization Results

==> Number Of Iterations     82
    Number Of Simulations    815

    Merit Function      4.3858634e+00
    Optimality               3.71e-06
    Feasibility              4.71e-11

==> Final Diagnostics: status code is 0

    The optimization seems to be successful.
    The required accuracy has been achieved.


    Final running mode: normal


==> End Of Optimization.
```

Small values of the optimality and feasibility numbers indicate that an optimum has been found. A more detailed analysis can be performed using the results printed in the *.otm* file.

The .otm file and the viewotm script

The optimization file is a structured text file that is separated into sections and subsections. It uses a technique known as folding, which allows the user to filter out one or more sections in the *.otm* file, providing an overview of the results. Like a piece of paper which is folded to make it shorter. The advantage of folding is that the user can get a better overview of the structure of text, by folding lines of a section and omitting it. This technique is based on a similar feature available in the text editor `vim`.

The *.otm* file is then formatted for that purpose. The user has to run the service routine named `viewotm` on the *.otm* file in order to display the results. This routine can be invoked as follows:

`viewotm` `[-l d] -f file.otm`

- `-l d`

  Where `d` is the level of detail that users wants to view. The range of levels are 1 to 5, level 1 (default) gives the least amount of data while level 5 gives the most amount of data.

  - Level `1`: only the results of the optimization are printed (merit function, values of the variables, and the objectives)
  - Level `2`: some additional informations related to the initialization phase are printed.

- o Level `3`, `4`, or `5` the results and data is more detailed.

When an outer loop has been performed, the amount of data can be very large. In this case level `1` only reports simplified results in a table printed at the end of the *.otm* file. An example is given in "Additional experiments" on page 1153.

- `-f`

  Name of the *.otm* file to be formatted.

For example, typing only: `viewotm -f lnaopt.otm` for the LNA problem will give the following output which are the final results (some very long lines were truncated to fit the width of the page). The sections concerning the scaling of variables and objectives can be viewed at higher level of details.

```
Optimization Results
====================

 Section 1 - Statistics and Diagnostics
 --------------------------------------

 ==> Number Of Iterations     83
     Number Of Simulations    838

     Merit Function      4.3858634e+00
     Optimality               2.01e-05
     Feasibility              1.49e-09

 ==> Final Diagnostics: status code is 0

     The optimization seems to be successful.
     The required accuracy has been achieved.


 ==> Final running mode: normal

 ==> Global elapsed time:  47 seconds


 Section 2 - Status Of Variables
 -------------------------------


     Name          Status  Init Value       Final Value      Lower Bound ...

     LS            UPR      2.5000e-10       3.0000e-09       0.0000e+00 ...
     CPIN1         LWR      1.0000e-13       0.0000e+00       0.0000e+00 ...
     CPIN2         BND      1.0000e-13       2.2340e-13       0.0000e+00 ...
     CSIN          LWR      1.0000e-12       1.0000e-12       1.0000e-12 ...
     LSIN          BND      1.0000e-10       1.0582e-08       0.0000e+00 ...
     N1            BND      3.0000e+01       1.9099e+01       1.0000e+01 ...


 Last Simulation
 ===============
```

```
==> Results of Last Simulation


*** Goal/Equal Objectives in simulator id(0)

    Name            Init Value     Curr Value     Goal Value     Weight
    AV_DB@2.4GHZ      1.1922e+01      2.0100e+01      2.0000e+01 1.0e+00

    AV_DB@2.5GHZ      1.1734e+01      2.0128e+01      2.0000e+01 1.0e+00


*** Inequality Constraints in simulator id(0)

    Name            Type  Feas  Init Value     Curr Value     Lower Bnd
    S11_DB@2.4GHZ    Range  <.>  -8.2277e-01     -1.4164e+01 -1.5000e+01

    S11_DB@2.5GHZ    Range  <.>  -8.7893e-01     -1.5000e+01 -1.5000e+01


*** Minimize/Maximize Objectives in simulator id(0)

    Name             Init Value      Curr Value      Weight    Rel Chg %
    NF_DB@2.4GHZ      3.0310e+00      1.0860e+00     1.0e+00   -64.2

    NF_DB@2.5GHZ      3.0631e+00      1.1232e+00     1.0e+00   -63.3

    NFMIN_DB@2.4GHZ   1.0165e+00      1.0517e+00     1.0e+00    3.5

    NFMIN_DB@2.5GHZ   1.0640e+00      1.1118e+00     1.0e+00    4.5


==> End Of Optimization
```

When one outer loop is performed, the lowest level of details is represented by a table of results such as follows. There is one line for each optimization run.

```
Optimization Results With Respect To The Outer Loop
 ===================================================

  Run   Iter  Simul   Merit Function Optim Outer Parameters

     1    22   177     5.0741859e+01 6.4e-04 LS: 0.0000e+00  N1: 1.0000e+01
     2    11    97     6.7559608e+01 1.5e-03 LS: 0.0000e+00  N1: 1.5000e+01
     3    26   205     6.2095585e+01 2.3e-04 LS: 0.0000e+00  N1: 2.0000e+01
     4    33   274     5.2472654e+01 1.1e-01 LS: 0.0000e+00  N1: 2.5000e+01
     5    31   242     4.1993472e+01 9.3e-02 LS: 0.0000e+00  N1: 3.0000e+01
     6    15   130     1.2913451e+02 9.9e-05 LS: 0.0000e+00  N1: 3.5000e+01
...
```

How to monitor the design objectives (MONIT_VAL options)

The information in the *.otm* file can be enriched. Some additional columns labelled with the name of a design objective can be printed during the optimization process. Note that when a design objective is represented by a single value, we will print this value as a “raw number”.

The purpose of this feature is to allow you to monitor the optimization process. Consider the following case:

```
Iter Simul LS Step  Merit Function   Feas    Optim    Slack    FIT_IDRAIN
  1   10  1.0e+00   2.6318512e+01  0.0e+00          0.0e+00  4.1230176e-01
  2   20  1.0e+00   2.5609867e+01  0.0e+00  2.0e+00  0.0e+00  3.7050156e-02
  3   39  3.0e+00   1.6093292e-01  0.0e+00  2.0e+00  0.0e+00 2.9060346e-02
  4   51  3.5e-02   1.6059068e-01  0.0e+00  6.3e-01  0.0e+00 2.7052106e-02
  5   63  2.1e+01   8.4372205e-02  0.0e+00  5.4e-01  0.0e+00 2.7050156e-02
  6   73  1.0e+00   8.0979363e-02  0.0e+00  8.1e-01  0.0e+00 2.7050007e-02
  7   85  2.6e+01   6.8441845e-02  0.0e+00  1.8e-01  0.0e+00 2.7050000e-02
...
```

The associated optimization statement could be:

```
.OBJECTIVE DC LABEL=FIT_IDRAIN I(V101)
+ GOAL=DATA_D25X25(IDS) TYPVAL=1E-4 MONIT_VAL=ERRMEAN
```

Usually we do not print each error function as only the global error function, also known as RMS function, is needed. This is the sum of the squared residuals:

$$\frac{1}{N_r}\sum_k (f_r^{(k)}(x) - r^{(k)})^2$$

where: $r^{(k)}$is the goal value associated to the *k*th error function, and N_r the number of error functions.

We can also print the maximum or the minimum value for the whole set of values

$$\left\{f_r^{(1)}(x),\ \ldots,\ f_r^{(N_r)}(x)\right\}$$

These options are explained in the following list:

- **MONIT_VAL=MEAN**, the printed value will be the sum $\frac{1}{N_r}\sum_k f_r^{(k)}(x)$ (which is the mean value of the functions $f_r^{(k)}(x)$),
- **MONIT_VAL=MAX** or **MONIT_VAL=MIN**, the printed value will be the maximum $Max\{f_r^{(k)}(x)\}$ or the minimum $Min\{f_r^{(k)}(x)\}$,
- **MONIT_VAL=ERRMEAN**, the printed value will be the sum $\frac{1}{N_r}\sum_k (f_r^{(k)}(x) - r^{(k)})^2$ of the squared residuals,
- **MONIT_VAL=ERRMAX** or **MONIT_VAL=ERRMIN**, the printed value will be the maximal error function $Max\{\left|f_r^{(k)}(x) - r^{(k)}\right|\}$ or the minimal error function $Min\{\left|f_r^{(k)}(x) - r^{(k)}\right|\}$.

When the design objectives are specified as **MINIMIZE** and **MAXIMIZE** the situation is similar, but there is no notion of “error function” associated to these objects. One can consider the following netlist as a basic example:

```
* Design parameter (temperature)
.STEP TEMP -25 150 5

* Design variable specification
.PARAMOPT
+ x = (x0,a,b)

* Minimization
.OBJECTIVE
+ EXTRACT_INFO LABEL=f_m
+ {$MACRO|FUNCTION}
+ GOAL=MINIMIZE
+ PRN_VAL=ERRMEAN
```

This will minimize the functions $f_m(x;T_k)$ with respect to x subject to the constraints $a \le x \le b$, where T_k represents a point of discretization in the temperature interval [-25, 150]. It could be useful to monitor the mean value or the max/min values of these objectives. These options are addressed in the following list:

- **MONIT_VAL=MEAN**, the printed value will be the sum $\frac{1}{N_m}\sum_k f_m^{(k)}(x)$,
- **MONIT_VAL=MAX** or **MONIT_VAL=MIN**, the printed value will be the maximal value $Max\{f_m^{(k)}(x)\}$ or the minimal value $Min\{f_m^{(k)}(x)\}$.

We now consider the case of inequality constraints. The notion of “error functions” can be extended to this case, we call these functions the *violation* associated to a constraint. We will consider a group of range constraints:

$$l^{(k)} \le f_i^{(k)}(x) \le u^{(k)},$$

the maximum violations v_u and v_l are defined as follows:

$$v_u^{(k)} = max\{0, f_i^{(k)}(x) - u^{(k)}\}$$

and

$$v_l^{(k)} = max\{0, l - f_i^{(k)}(x)\}$$

These numbers are always positive $v_{1,u} \ge 0$. This is summarized in the following list:

- **MONIT_VAL=MEAN**, the printed value will be the sum $\frac{1}{N_i}\sum_k f_i^{(k)}(x)$,
- **MONIT_VAL=MAX** or **MONIT_VAL=MIN**, the printed value will be the maximal value $Max\{f_i^{(k)}(x)\}$ or the minimal value $Min\{f_i^{(k)}(x)\}$,

- **MONIT_VAL=ERRMEAN**, the printed value will be the mean value of the constraint violations $\frac{1}{N_i}\sum_k(\nu_l^{(k)} + \nu_u^{(k)})$,
- **MONIT_VAL=ERRMAX** or **MONIT_VAL=ERRMIN**, the printed value will be the maximal constraint violation $Max_k\{\nu_l^{(k)} + \nu_u^{(k)}\}$ or the minimal constraint violation $Min_k\{\nu_l^{(k)} + \nu_u^{(k)}\}$.

Final diagnostic conditions

When the optimizer has finished, a `status code` is printed in the `Optimization Results`. It is essential to check that the value of the return code (and its associated message) and the effective results of optimization (design variables and objectives). The final diagnostics only indicate the final status of the optimizer before it exits. These messages and their meaning are described in Table 18-11:

Table 18-11. Optimization Result Status Code

Code	Message and meaning
0	`The optimization seems to be successful. The required accuracy has been achieved.` The iterates have converged to a point $\bar{x}$ that satisfies the optimality conditions to the accuracy requested by the optional parameters **TOL_OPT** or **TOL_GRAD** and **TOL_FEAS** (please refer to “The Optimization Methods” on page 1185). The user should verify whether the following two conditions have been satisfied: o The final value of `Optimality` is significantly small. o The final values of `Feasibility` is significantly small. Note, these values are set to zero if there are no equality or inequality constraints.
1	`Optimization was not successful. The maximum number of simulation runs has been reached.` The limiting number of iterations, determined by the optional parameter **MAX_ITER** has been reached. Check the iteration log contained in the *.otm* file. If the optimizer appears to be making progress, **MAX_ITER** may be too small. If so, increase its value and rerun the optimizer, possibly using the values of the design variables obtained so far (this can be considered as a *warm start*).

Table 18-11. Optimization Result Status Code

Code	Message and meaning
2	`The current point cannot be improved. A sufficient decrease in the merit function could not be attained during the last line search. This may occur because the user has requested an overly stringent accuracy.` A sufficient decrease in the merit function could not be attained during the final line search. This sometimes occurs because an overly stringent accuracy has been requested, i.e. **TOL_OPT** is too small. In this case the user should verify the two conditions described under Code = 0 to determine whether or not the final solution is acceptable. It may happen that an additional status code is raised. This status can only take the values: Code= 0, Code=6, or Code=11.
4	`Optimization was not successful. The maximum number of simulation runs has been reached.` The limiting number of simulations, determined by the optional parameter **MAX_SIMUL** has been reached.
6	`The current point is feasible and quite optimal, however, the required accuracy has not been achieved. The optimizer is within 1e-2 of satisfying the TOL_OPT (or TOL_GRAD) optimality tolerance.` The optimizer is within $1x10^{-2}$ of satisfying the **TOL_OPT** (or **TOL_GRAD**) optimality tolerance. For next runs, the user should check that **TOL_OPT** is not too small.
7	`The objective function may be unbounded on the feasible set. Alternatively, it may happen that the scaling of the problem is very poor.` To avoid this situation, try to define bounds on some variables if the **MINIMIZE** or **MAXIMIZE** objectives are close to singularities. Also use an appropriate scaling of the design objectives.
8	`The simulation process stopped the optimization. The optimization was abandoned.` This message is raised only after an interrupt message (Ctrl+C).
9	`The current point cannot be improved. The problem is too hard to solve. Try with another starting point.` This has to be considered as a failure of the optimizer regarding the treatment of constraint feasibilities. Please refer to “Normal and elastic modes of termination” on page 1200. Check that the final solution is not an acceptable point for the problem.

Table 18-11. Optimization Result Status Code

Code	Message and meaning
10	`The current point cannot be improved. The Hessian matrix has been reset too many times. It indicates that the line search failed to find a sufficiently better point for a large number of iterations.` This message should appear very rarely. This is an algorithmic weakness of the method implemented in the Eldo optimizer/SQP. Try to start the optimization from a different point.
11	`The starting point cannot be improved. It may happen that the derivative are inaccurate or the starting point is already critical.` Check that the objectives to optimize are well-defined at the starting point.
12	`The current point cannot be improved. The last QP step was too poor to allow a sufficient progress of the iterations.` This may indicate that the extracted measures and their derivatives have a low precision at the current point.
13	`Optimization was not successful. Either the objective function or the constraints seem to be undefined at the starting point. The optimization was abandoned.` The current version of the Eldo optimizer/SQP cannot treat problems where the objectives returned by Eldo (at the first iterate) have the UNDEF value. Try to eliminate such cases by using a two-stage approach.
14	Please refer to code 2.
15	Please refer to code 2.

Normal and elastic modes of termination

The SQP algorithm is able to make explicit allowance for infeasible constraints when computing the incremental steps obtained from a sequence of sub-problems. This phenomenon is referred to as local infeasibility.

In constrained optimization, a situation where no feasible solution exists can occur. In this case the constraints are inconsistent and the problem is infeasible. If all the constraints are bounds on the variables:

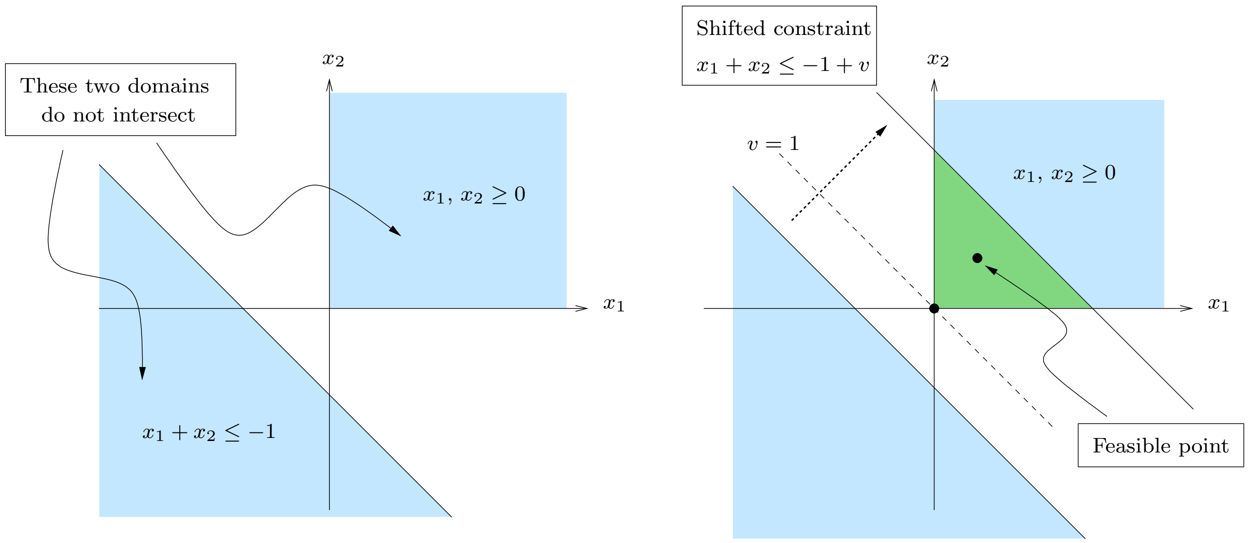
$$x_l^{(i)} \leq x^{(i)} \leq x_u^{(i)}$$

It is simple to determine whether a feasible point exists, since the *i*th is only inconsistent when:

$$x_l^{(i)} > x_u^{(i)}$$

Such infeasibilities are automatically trapped when parsing the **.PARAMOPT** command. Conversely, with general constraints there are no simple characteristics that identify whether a feasible solution exists.

Figure 18-14. Illustration of the ‘elastic’ mode (1)



The following simple example illustrates the possible situation, see also the figure 18-14 for a graphical illustration. Consider the optimization problem with two variables X1 and X2

```
.PARAMOPT X1=(4, 0, *)
.PARAMOPT X2=(7, 0, *)

.OBJECTIVE DC LABEL=C (X1 + X2) UBOUND=-1
```

These statements define two bound constraints on the variables: $x_1 \geq 0$ and $x_2 \geq 0$, and the linear constraint $x_1 + x_2 \leq -1$. By considering the graphical representation, one may also see that they define two disjoint sets of points. The optimization problem is then infeasible.

In order to cope with this issue, the SQP optimizer can run in one of two different modes:

- *normal* mode
- *elastic* mode

The algorithm automatically enters the ‘elastic’ mode when a situation of infeasibility is detected. This mode performs a shift of the bounds on constraints. In our example, we can add a positive number $v \geq 0$ to the upper bound **UBOUND**:

$$x_1 + x_2 \leq -1 + v$$

and try to minimize the value of v or minimize the *constraint violation*. This numerical modification is graphically displayed in the right part of figure 18-14, where the green colored area represents the non empty feasible domain. It is clear that $v = 1$ is the smallest of the violation such that there exist a feasible point.

For example, we solved this simple problem with the SQP optimizer. The *.otm* file gives the following results for the optimized variable:

```
Section 2 - Status Of Variables
-------------------------------

     Name          Status  Init Value      Final Value      Lower Bound

     X1            BND      4.0000e+00      0.0000e+00       0.0000e+00
     X2            BND      7.0000e+00      0.0000e+00       0.0000e+00
```

where the final value of the parameters is (0,0) the origin point. The final status of the constraint is given by

```
Section 3 - Status Of Objectives
--------------------------------

 *** Inequality Constraints in simulator id(0)

 Type    Name              Init Value      Opt Value       Violation
 UPR     CONSTR1       *** 1.1000e+01       1.0000e-06  *** 1.0000e+00
```

The optimal value (`Opt Value`) of the constraint is 1.0×10^{-6} and the number in the last column (`Violation`) gives the violation or equivalently the value of the additional variable *v*. In this case, the violation is a strictly positive value 1.0.

Note that this approach has very interesting properties, for instance the optimal solution often violates only a small number of the constraints. Therefore, a point that satisfies many of the constraints has been computed, i.e. a large subset of constraints that are feasible have been identified. This is more informative for the user, than finding that the constraints are mutually feasible (this approach is closely related to the L1-norm regularization technique).

Consider the following problem with two variables. We want to minimize some function $f(x_1, x_2)$ subject to the constraint

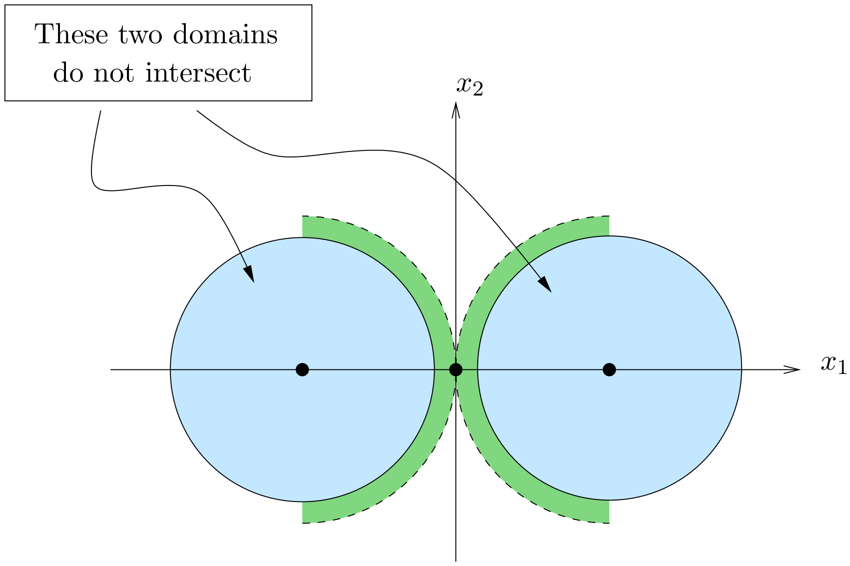
$$(x_1 + 1)^2 + x_2^2 \le \frac{3}{4}$$

which defines a disk centered at (-1,0) with radius $\sqrt{3}/2$, and the second constraint

$$(x_1 - 1)^2 + x_2^2 \le \frac{3}{4}$$

which defines the symmetrical disk centered at (1,0). This situation is displayed in figure 18-15.

Figure 18-15. Illustration of the ‘elastic’ mode (2)



It is clear that there is no feasible solution. If we run the Eldo optimizer on this example:

```
.PARAMOPT X1=(-2, *, *)
.PARAMOPT X2=(-2, *, *)

.OBJECTIVE DC LABEL=CIRCLE1 ( (X1-1)**2 + X2**2 ) UBOUND=0.75
.OBJECTIVE DC LABEL=CIRCLE2 ( (X1+1)**2 + X2**2 ) UBOUND=0.75
```

we get the following output

```
==> Starting Optimization...

 Iteration 0:  Feas = 1.2e+01,  Optim = 6.0e+00

 Iter Simul    LS Step   Merit Function    Feas     Optim     Slack

    1    10    1.0e+00    6.3505943e-01  1.2e+00  6.0e-04  0.0e+00
    2    16    3.3e-01    1.7787634e+00  3.9e-01  7.8e-01  0.0e+00
    3    22    1.9e-01    4.3108310e+00  2.9e-01  9.0e-01  0.0e+00
    4    28    3.4e-02    3.7393098e+01  2.5e-01  1.3e+00  0.0e+00
    5    34    3.7e-04    6.5283185e+03  2.5e-01  1.9e+00  0.0e+00
    6    40    1.0e-00    1.5304677e+04  2.5e-01  2.0e+00  3.5e+00
    7    46    2.2e-01    1.5238534e+04  5.1e-03  7.7e+03  3.6e-01
    8    53    1.3e-02    1.4929878e+04  5.1e-03  7.5e+03  3.5e-01
    9    61    9.9e-01    1.5000628e+04  1.1e-02  1.9e+03  3.5e-01

 Iter Simul    LS Step   Merit Function    Feas     Optim     Slack

   10    65    1.0e+00    1.5000626e+04  4.1e-04  2.8e+01  3.5e-01
   11    69    1.0e+00    1.5000625e+04  1.2e-07  1.4e+01  3.5e-01

==> End Of The Optimization Phase
```

The column `Slack` is the value of the global constraint violation, it is the norm of the artificial variables (v_1, v_2) added to the upper bound **UBOUND** for each constraint. For instance the first constraint becomes

$$(x_1 + 1)^2 + x_2^2 \le \frac{3}{4} + v_1$$

and similarly the second constraint becomes

$$(x_1 - 1)^2 + x_2^2 \le \frac{3}{4} + v_2$$

The value `Slack` becomes strictly positive after the 6th iteration. This phenomenon can be interpret as follows:

- after the 6th iteration the algorithm detected that the problem is locally infeasible,
- two artificial variables are then introduced such that the constraints become consistent. Graphically, this can be interpreted as increasing the radius of the two circles such that they do intersect.
- The goal of the algorithm is then to find a solution point such that the constraint violation is minimized.

The final solution is given in the following output:

```
Section 2 - Status Of Variables
-------------------------------

    Name          Status  Init Value      Final Value

    X1            FR      -2.0000e+00     -5.0009e-07
    X2            FR      -2.0000e+00      1.6166e-05


Section 3 - Status Of Objectives
--------------------------------

*** Inequality Constraints in simulator id(0)

Type    Name             Init Value      Opt Value        Violation
UPR     CIRCLE1      *** 1.3000e+01       1.0000e+00  *** 2.5000e-01

UPR     CIRCLE2      *** 5.0000e+00       1.0000e-00  *** 2.5000e-01
```

We can observe that the constraint violations have been minimized. The actual value of the violation is $v_1 = 1/4$ for the first constraint and $v_2 = 1/4$ for the second. The solution point is approximately the origin (0,0). These remarks are consistent. The algorithm automatically increased the radius of each circle such that the feasible domain becomes a non empty set. The feasible set reduces to the singleton (0,0).

Final status of variables

The final value of the design variables are listed in the second section of the optimization results file (*.otm*). This section starts with the title: "`Section 2 - Status Of Variables`".

The output informations are the initial and final continuous value of the design parameters labeled with "`Init Value`" and "`Final Value`" respectively. The lower and upper bounds are also printed. The value labeled with "`Discr Value`" represents the final value after projection onto the grid of discretization (consult "Discretization of Design Variables" on page 1146 for details).

Final status of objectives

The final value of the objectives are listed in the third section. This section starts with the title "`Section 3- Status Of Objectives`". The objectives are printed according to their category. The level of the output is controlled by the argument **`PRNVAL`** (see ".OBJECTIVE" on page 1134). The printed values are described below:

- When the design objective has the **`GOAL`** (and **`EQUAL`**) argument. The initial, the current and goal values are printed with the labels: "`Init Value`", "Opt `Value`" and "`Goal Value`" respectively. The value "`Rel Chg`" represents the relative change (defined in percentage) of the design objectives. The weight number is also printed (as a percentage).
- The **`MINIMIZE`** and **`MAXIMIZE`** objectives are printed with their initial and final value. The value "`Rel Chg`" represents the relative change of the objective.
- The inequality constraints (with **`LBOUND`** and **`UBOUND`** arguments) are printed with an additional status represented by a symbol "`***`" if the constraint violation is non zero. This symbol is printed to indicate that the initial or the final objective are not satisfied.

Last simulation with final solution

The results of optimization are compared to the final results obtained after discretization of the design variables. This section starts with the title "`Analysis Of Last Simulation`". The objectives are printed according to their category. For example, the 'optimized value' of a **`MINIMIZE`** objective is labeled as "`Opt Value`" and the final value obtained after discretization is given in the column "`Discr Value`". It is then possible to observe a possible discrepancy due to the discretization algorithm.

References

1. J.F. Bonnans, J. Ch. Gilbert, C. Lemarechal, C. Sagastizabal. *Numerical Optimization -Theoretical and Practical Aspects*. Springer Verlag, Berlin, 2003.
2. R. Fletcher. *A general quadratic programming algorithm*. Journal of the Institute of Mathematics and its Application, 7, 76-91, 1971.

3. R. Fletcher. *Practical Methods of Optimization* (second edition). John Wiley & Sons, Chichester, 1987.
4. P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, 1981.
5. J. Nocedal and S. W. Wright. *Numerical Optimization*. Springer Series in operations Research, Springer, New York, 1999.
6. R. Point, M. Mendes and W. Foley. *A differential 2.4GHz Switched-Gain CMOS LNA for 802.11b and Bluetooth*. 2002 IEEE Radio and Wireless Conference.

Chapter 19
Monte Carlo Analysis

Introduction

Originally, the Computer Aided Design (CAD) tools have been used to study the nominal design of an integrated circuit (IC). Due to the disturbances of the IC manufacturing process, the effective performance of the mass produced chips are different than those for the nominal design. Process-related performance variations may lead to low manufacturing yield, and unacceptable product quality. For these reasons, statistical circuit design techniques are required to design the circuit parameters.

The purpose of the Monte Carlo (MC) analysis is to determine the uncertainty in estimates for dependent variables of interest. Thus MC analysis focuses on data, and *how uncertainty in data propagates* through computations. This definition of uncertainty involves model input and output models. In our context, the models of interest are provided by circuit simulations. For instance, Eldo takes a netlist describing the circuit transistors, resistors, capacitors, and so on, and their connections and translates this description into mathematical equations. The inputs are therefore the various design parameters, the process parameters and the environmental conditions. On the other side, the output space is characterized by the circuit performance of interest.

Monte Carlo-based uncertainty analysis is performed on multiple model evaluations with randomly selected model input variables, and then using the results of these evaluations to determine the uncertainty in model predictions, and the input variables that gave rise to this uncertainty. In general, a Monte Carlo analysis involves four steps:

- A range and distribution are selected for each input factor. These selections will be used in the next step in the generation of a sample from the input factors.
- A sample of points is generated from the distribution of the inputs specified in the first step. The result of this step is a sequence of sample elements.
- The circuit simulator is fed with the sample elements and a set of extracted measures is produced. In essence, these evaluations create a mapping from the space of the inputs to the space of the results. This mapping is the basis for subsequent uncertainty analysis.
- The results of model evaluations are used as the basis for the uncertainty analysis. For example, one way to characterize the uncertainty is with a mean value and a variance. Other output statistics are also provided.

The Monte Carlo-based approach has several advantages: it forces explicit acknowledgment of all sources of uncertainty, it can take account of any distribution and correlation and can be applied to complex simulations.

Usage

Define the .MC Command

The first step in Monte Carlo is to define the **.MC** command.

```
.MC RUNNO [OUTER] [OV] [SEED=integer_value] [NONOM] [ALL]
+ [VARY=LOT|DEV] [IRUN=val] [NBBINS=val] [ORDMCS] [MCLIMIT]
+ [PRINT_EXTRACT=NOMINAL|ALL|run_number] [SIGBIN=val]
+ [MAXABSBIN=val] [MAXRELBIN=val]
+ [MONITOR] [AUTOSTOP=expression]
+ [SAVE=mc_file] [RESTART=mc_file]
```

- RUNNO

  Number of simulation runs (probability samples). Integer.

- **OUTER**

  When there are both **.STEP** and **.MC** commands, Eldo performs a full Monte Carlo analysis for each point of the **.STEP** command. If the keyword **OUTER** is specified on the **.MC** command the nesting of the simulations will be inverted. Instead, a **.STEP** will be performed at each Monte Carlo run (the **OUTER** keyword must be placed after the specification of the number of Monte Carlo runs). **.MC OUTER** does not work with a variation in temperature with the **.TEMP** command (for example **.TEMP** 0 39 80) or **.STEP TEMP** command, but does work with a single temperature definition with **.TEMP** (for example **.TEMP** 70).

- OV

  Compute the standard deviation of the quantity OV and output to the ASCII output file. The format of OV is one of:

  I(Vxx[, Vyy]), which specifies the difference in current between voltage sources Vxx and Vyy. If Vyy and the comma are omitted, then this will return just the current through Vxx;

  V(n1[, n2]), which specifies the voltage potential between nodes n1 and n2. If n2 and the comma are omitted, then this will return the potential between node n1 and ground.

  Other analysis output formats are available for AC analysis. Please refer to “.PRINT” on page 830.

- **SEED**=i

  The integer i is used to initialize the pseudo-random sequence of numbers for the probability distribution. Running Monte Carlo twice with the same seed value will give the same results.

- **NONOM**

  Nominal run for Monte Carlo analysis is bypassed. This option is used by Accusim. When **NONOM** is active, the **ALL** option of **.MC** is enabled as well.

- **ALL**

  When specified, the waveform results of every Monte Carlo simulation run are stored in the output files (one set per runno). Without this, only the nominal, minimum, and maximum results are saved. This includes both ASCII and binary wave results.

- **IRUN**=n

  n is an integer. This runs only the nth Monte Carlo simulation of a series. For example:

  ```
  .MC 10 IRUN=3
  ```

  tells Eldo to run the third Monte Carlo analysis of the 10 point series. If **IRUN** ≤ 0 it will be ignored and a warning generated.

  This can also be used with simulation information that has been saved for a specific run (index) of a Monte Carlo analysis using the **CARLO** argument of the **.SAVE** command. See “.SAVE” on page 857. This is useful for debugging purposes, as the DC value would be re-injected into the single Monte Carlo run specified.

- **VARY**=**LOT**|**DEV**

  Determines whether Monte Carlo variations are independent or correlated for model Monte Carlo variables. When specifying DEV, then DEV and DEVX variation is taken into account. Only one VARY specification can be set. By default, Eldo applies LOT, DEV and DEVX variation.

- **NBBINS**=VAL

  Specifies the number of bins for the histogram produced when Monte Carlo analysis is used with **.EXTRACT** statements. The default is 10.

- **ORDMCS**

  Determines whether multiple Monte Carlo parameters in the simulation share the same pseudo-random probability values or not.

  For example, without ORDMCS, Eldo generates a single probability stream p1, p2, p3, …, pn. Each probability is between 0 and 1. These probability series are shared among the variables. For two Monte Carlo variables, A and B, the assignment is:

  Run 1: A = p1, B = p2
  Run 2: A = p3, B = p4
  Run 3: A = p5, B = p6
  Run n: A = p[2n-1], B = p[2n]

  For four Monte Carlo variables, A, B, C, and D, the assignment is:

  Run 1: A = p1, B = p2, C = p3, D = p4
  Run 2: A = p5, B = p6, C = p7, D = p8
  Run 3: A = p9, B = p10, C = p11, D = p12
  Run n: A = p[4n-3], B=[4n-2], C=[4n-1], D=[4n]

  With ORDMCS, each variable gets its own probability series. For two Monte Carlo variables, the assignment is:

Run 1: A = pa1, B = pb1
Run 2: A = pa2, B = pb2
Run 3: A = pa3, B = pb3
Run n: A = pa[n], B = pb[n]

For four Monte Carlo variables, the assignment is:

Run 1: A = pa1, B = pb1, C = pc1, D = pd1
Run 2: A = pa2, B = pb2, C = pc2, D = pd2
Run 3: A = pa3, B = pb3, C = pc3, D = pd3
Run n: A = pa[n], B = pb[n], C = pc[n], D = pd[n]

- **MCLIMIT**

  Specifies that all parameters with statistical distribution (**DEV**, **DEVX**, or **LOT**) will have their distribution modified to one of two deviation values. These values correspond to the maximum deviation of the original distribution as defined by the option **SIGTAIL**. For example, a parameter with a nominal value of 1.0, a statistical deviation of `DEV/GAUSS=5%`, and with option **SIGTAIL** at its default value of 4, the two values will be calculated as follows:

  ```
  1-(4*0.05)=0.8,    1+(4*0.05)=1.2
  ```

  This functionality can be useful to force Monte Carlo runs to use maximum deviation combinations. **MCLIMIT** affects all statistical parameters whatever their original distribution.

- **PRINT_EXTRACT**=**NOMINAL|ALL|**run_number

  Specifies for which run extracted values should be printed and written to output files.

  NOMINAL

  Only the nominal extracted value is printed (default).

  ALL

  Extracted values are printed for all runs.

  run_number

  Extracted values are printed only for the specified run_number (0 is the nominal run).

- **SIGBIN**=VAL

  Can be specified to truncate the histogram to a certain number of sigmas. Eldo will gather all the samples above "mean+SIGBIN×sigma" in the Above bin, and all the samples below "mean−SIGBIN ×sigma" in the Below bin. This might be useful when there are a few untypical samples that would otherwise corrupt the min and max of the histogram.

- **MAXABSBIN**=VAL

  Can be specified to simplify the histograms printed out in the .chi file, if they are difficult to read if most of the samples are in the same bin. If there are more than MAXABSBIN terms in a bin, then that bin will be expanded recursively, and a new histogram will be printed out for that bin. This corresponds to a zoom in each of the bins which contain too much samples. Default value is -1, i.e. this option is not active by default.

- **MAXRELBIN**=VAL

  Can be specified to simplify the histograms printed out in the .chi file, if they are difficult to read if most of the samples are in the same bin. If there are more than MAXRELBIN% of the samples in the same bin, then that bin will be expanded recursively, and a new histogram will be printed out for that bin. This corresponds to a zoom in each of the bins which contain too much samples. Default value is -1, i.e. this option is not active by default.

- **MONITOR**

  Monitor the evolution of certain quantities. Eldo will flush out from the *.wdb* file the average and standard deviation of extracts (**.EXTRACT**) and measurements (**.MEAS**) versus the Monte Carlo run index in order to see how these entities evolve. Eldo will also flush out of the *.wdb* file the expressions used in the **AUTOSTOP** criteria.

- **AUTOSTOP**=expression

  Automatically stops the Monte Carlo process based on the convergence of some or all of the quantities defined in the expression. The expression in the **AUTOSTOP** clause is a boolean expression using the **MCCONV** extracts as described in "Monte Carlo Convergence" on page 1218.

- **SAVE**=mc_file

  Saves any relevant information to a specific file. This files can then be used start a new session inheriting the results from the saved session. This save feature is disabled when multiple run commands are specified (**.STEP**, multiple **.TEMP**, **.AGE**, and **.MPRUN**).

- **RESTART**=mc_file

  Restarts a Monte Carlo simulation run from a previous session on the same design saved with the **.MC ... SAVE**=mc_file specification. If the file, mc_file, does not exist Eldo generates a warning and performs a normal Monte Carlo run. If a restart file is specified, it means that the number of runs indicated on the command is the additional number of runs.

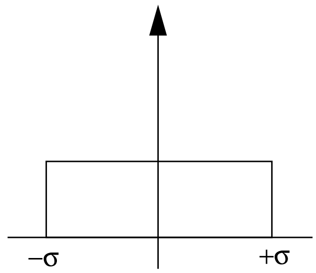
  Topology or stimuli condition changes are allowed, but are not meaningful. The histogram and Monte Carlo statistics reported at the end of an incremental Monte Carlo "session" are computed using data from the previous session. The waveforms displayed versus the run index display the total information, that is, combine the successive runs.

Assign Nominal Parameter Values

In the second step, each Monte Carlo parameter is assigned a nominal value, a standard deviation (expressed in an absolute value or a relative percentage of the nominal), and LOT/DEV correlation. These are the available probability functions:

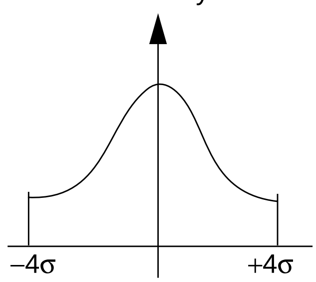
Uniform

Uniform probability, as it sounds, spreads the probability evenly over the sample space. The probability of a sample occurring outside of σ of the nominal value is zero. This is the default probability function.



Gaussian

A Gaussian probability function resembles a bell curve. In Eldo, the curve is truncated at 4σ. Note that the probability of a sample landing σ of nominal is 68.3%, and within 3σ of nominal is 99.99%.



User-defined

An arbitrary model can be defined through the **`.DISTRIB`** command. Please refer to “.DISTRIB” on page 614 for more details.

LOT/DEV Correlation

Monte Carlo parameters can be correlated or not. For example, capacitors in an IC design may have correlation, for example their capacitance values tend to rise or fall together. This is LOT correlation. However, components on a printed circuit board tend not to be correlated, which is DEV correlation. LOT causes devices to vary with each other. DEV causes devices to vary independently of each other.

Parameters can have both LOT and DEV variation. Using both means that affected devices have a degree of independence, even though they are correlated. If a parameter is not a primitive, but depends on parameters with no LOT/DEV specification, then a LOT /DEV specification can be set on that parameter.

A 20% LOT setting with a 5% DEV setting for a uniform distribution means that the parameter may not exceed 25% off of nominal. Furthermore, devices that share that parameter are assigned the 20% LOT variation together (s1); then, individual 5% DEV variations are assigned based on s1, instead of the nominal value.

Note

DEV variation specified with **`.MODEL`** statements (or **`.MCMOD`**) can refer to dimensions of the current object directly, without any need to encapsulate models into subcircuits. The syntax for accessing an instance parameter is for example:

```
E(*,<instance_parameter_name>)
```

The * character is used here to refer to the current instance.

A LOTGROUP can be defined to share the same distribution between dissimilar elements. Once a LOTGROUP is defined, it is used in the same way as LOT or DEV. See the Define Monte Carlo Parameters section below for an example.

```
.LOTGROUP group_name[/distrib_type]=val[%]
```

Define Monte Carlo Parameters

Monte Carlo parameters are defined through **`.MODEL`**, **`.MODEL`** with **`.MCMOD`**, or **`.PARAM`** statements.

.MODEL

Individual model parameters can be given different probability functions. To invoke this, a DEV and/or LOT parameter must immediately follow the model parameter. For example:

```
.model n nmos vto=0.6 dev=0.4 tox=1.5e-7
```

defines a NMOS model with `vto` set to 0.6V and `tox` set to 1.5e-7. The model parameter `vto` has a uniform distribution with σ=0.4V, but `tox` is a constant. The following specification is equivalent:

```
.model n nmos vto=0.6 dev=66.67% tox=1.5e-7
```

In order to use a Gaussian distribution, the DEV parameter is changed to DEV/GAUSS. Using the same example, changing the `vto` distribution to Gaussian would alter the **`.MODEL`** card as follows:

```
.model n nmos vto=0.6 dev/gauss=0.4 tox=1.5e-7
```

If a **`.LOTGROUP`** is defined, then that can be used, as well. The Gaussian distribution example can be expressed as:

```
.LOTGROUP group_a/gauss=0.4
```

```
.model n nmos vto=0.6 lotgroup=group_a
.model n2 nmos vto=2.5 lotgroup=group_a
```

A different parameter of a different model using LOTGROUP group_a would have the same distribution. For example, the same number, between +0.4 and -0.4, will be used for both `vto` parameters.

Eldo takes into account changes in binning parameters (for example LMIN, LMAX) due to MC variation. Eldo will first make the variations on the **`.MODEL`**, and then select at each MC run the appropriate model. For example:

```
.param p1 = 1 dev = 5%
.MODEL N.1 NMOS LMIN = '1u*p1' LMAX = '2u*p1'
.MODEL N.2 NMOS LMIN = '2u*p1' LMAX = '3u*p1'
M1 D G S B N W=1u l = 1u
```

Here Eldo will extract a new random value for p1: this value will be used to update parameters of both N.1 and N.2. When all parameters of the **`.MODEL`** have been updated, Eldo will select the appropriate model to be used for device M1.

If both the size of a device and the binning parameters of its model are altered by MC variation, Eldo will issue an error that it cannot handle this situation.

Use option **`MC_IGNORE_BINNING`** to disable this automatic selection of the binning parameters at run time; the model selected at the nominal run will then be used for the whole MC process.

.MCMOD

Modifying **`.MODEL`** lines for Monte Carlo analysis can be inconvenient. Eldo offers the **`.MCMOD`** command to assign probability functions for **`.MODEL`** parameters without changing any part of the **`.MODEL`** statement.

```
.MCMOD model [(list_of_instances)] param1 LOT|DEV=val1
+ [{param2 LOT|DEV=val2 param3 LOT|DEV=val3 …}]
```

The **`.MODEL`** name must be specified, followed by one or more pairs of parameters and LOT and/or DEV values. Optionally, a list of instances will apply the Monte Carlo probabilities to only instances in that list.

Using the example above, the Monte Carlo parameter may be written as:

```
.model n nmos vto=0.6 tox=1.5e-7
.mcmod n vto dev=0.4
```

.PARAM

The syntax for Monte Carlo **`.PARAM`** parameters is different than for **`.MODEL`** parameters. There are two ways to use **`.PARAM`**. The simpler of the two ways is outlined; please refer to “.PARAM” on page 778 for additional details.

```
.PARAM param1=UNIF|AUNIF|GAUSS|AGAUSS|LIMIT({option_list})
```

- `UNIF(nom, rel)`

  Defines a uniform distribution. The parameter `param1` varies uniformly between `nom` - `nom`×`rel` and `nom` + `nom`×`rel`.

- `AUNIF(nom, abs)`

  Defines a uniform distribution. The parameter `param1` varies uniformly between `nom` - `abs` and `nom` + `abs`.

- `GAUSS(nom, rel, sigcoef)`

  Defines a Gaussian distribution. The curve is centered around `nom`, and the standard deviation is given by σ = (`nom` × `rel`) ÷ `sigcoef`.

- `AGAUSS(nom, abs, sigcoef)`

  Defines a Gaussian distribution. The curve is centered around `nom`, and the standard deviation is given by σ = `abs` ÷ `sigcoef`.

- `LIMIT(nom, abs)`

  Defines a limit distribution. Each sample can take the value of `nom` + `abs` or `nom` - `abs` only, with equal probability.

When defining a parameter using Monte Carlo distribution, variation on the parameter differs depending on where the parameter is specified. **`LOT`** variation is used when a defined parameter affects model parameters, this means the same random value will be used each time it affects a separate model parameter. **`DEV`** variation is used when a parameter affects instance parameters, an independent random value is calculated each time the parameter is specified.

Correlation

A correlation coefficient can be set up between parameters. Please refer to ".CORREL" on page 575 for details.

Monte Carlo Output

At the end of a Monte Carlo simulation, Eldo prints in the *.chi* file a histogram for each **`.EXTRACT`** command, see ".EXTRACT" on page 637. The number of bins for the histogram can be specified in the **`.MC`** command using the **`NBBINS`** parameter. The default is 10.

Note

The histogram is also output in the binary output file and can be displayed with EZwave.

The example below shows the *.chi* file entry for a Monte Carlo analysis (of least nine runs) with the following **`.EXTRACT`** command:

```
.EXTRACT tran label=tpd tpdud(v(in), v(out))
```

```
Distribution of TPD
```

```
                      Range [ 9.8405E-10  1.7374E-09]
                      Nominal value:  1.3124E-09
                      Average value:  1.2709E-09
                      Standard Deviation:  1.9007E-10
                      Standard Deviation based on nominal run:  1.3124E+00

   [ 984.00000P       1.06060N   ] NB = 2  FREQ = 1.00e+01% |*****
   [ 1.06060N         1.13720N   ] NB = 3  FREQ = 1.50e+01% |*******
   [ 1.13720N         1.21380N   ] NB = 5  FREQ = 2.50e+01% |************
   [ 1.21380N         1.29040N   ] NB = 1  FREQ = 5.00e+00% |**
   [ 1.29040N         1.36700N   ] NB = 4  FREQ = 2.00e+01% |**********
   [ 1.36700N         1.44360N   ] NB = 1  FREQ = 5.00e+00% |**
   [ 1.44360N         1.52020N   ] NB = 2  FREQ = 1.00e+01% |*****
   [ 1.52020N         1.59680N   ] NB = 1  FREQ = 5.00e+00% |**
   [ 1.59680N         1.67340N   ] NB = 0  FREQ = 0.00e+00% |
   [ 1.67340N         1.75000N   ] NB = 1  FREQ = 5.00e+00% |**
```

Two standard deviation results are provided:

- “Standard Deviation” is the RMS value of the deviation of output with respect to the average value;
- “Standard Deviation based on nominal run” is the RMS value of the deviation of output with respect to the nominal run.

A summary of the extracted Monte Carlo distribution results can be written to the *.aex* file and the file specified in the **.EXTRACT** command using the option **DUMP_MCINFO**, see “DUMP_MCINFO” on page 999 for more information.

In addition to the standard **.EXTRACT** arguments, see “.EXTRACT” on page 637, the following arguments specific to Monte Carlo analysis can be specified:

- **LBOUND**

  Lower bound of the value range
- **UBOUND**

  Upper bound of the value range

A Monte Carlo analysis can use this information to determine whether the extracted value remains in the range [**LBOUND**, **UBOUND**] during the Monte Carlo analysis, and display a report at the end of the ASCII output (*.chi*) file. For example:

```
.extract label=toto yval(v(s),25n) lbound=0.43 ubound=0.44
```

This may result in something like the below:

```
Distribution of TOTO
                  Range [ 420.84593M  461.95279M]
                  Nominal value:  440.36822M
                  Average value:  443.83611M
                  Standard Deviation:  12.80183M
                  Passed     :  4 ( 20.00000 %)
```

```
*** MC runs which passed all extract: 4 ( 20.0000 %)

 MC run 5 OK
 MC run 9 OK
 MC run 12 OK
 MC run 17 OK
```

Additional parameters can be specified on the **.EXTRACT** command to extract the minimum, maximum, mean and the standard deviation values, together with higher order moments:

- **MCMIN(**label**)**

  Returns the minimum value of the selected extract
- **MCMAX(**label**)**

  Returns the maximum value of the selected extract
- **MCNBECH(**label**)**

  Returns the number of measured values for the selected extract
- **MCAVG(**label**)**

  Returns the average value of the selected extract
- **MCSTD(**label**)**

  Returns the standard deviation of the selected extract
- **MCVAR(**label**)**

  Returns the variance of the selected extract
- **MCSKEW(**label**)**

  Returns the skewness of the selected extract, computed as the moment of order 3
- **MCKURT(**label**)**

  Returns the kurtosis of the selected extract, computed as the moment of order 4
- **MCMOM(**label,center_value,order**)**

  Returns for the selected extract, the moment of order centered on the value center_value

Note

Keyword **MC** is required on the **.EXTRACT** statements which make use of these Monte Carlo extract functions.

Examples:

```
.EXTRACT AC LABEL=myextract MAX(VDB(s))
.EXTRACT MC MCKURT(myextract)
.EXTRACT MC
+ MCMOM(myextract,MCAVG(myextract),4.0/MCSTD(myextract)**2)
```

The third line is equivalent to the **MCKURT** function.

Monte Carlo Convergence

Specify the **MCCONV** function to control the run-length in the context of sequential Monte Carlo algorithms for estimating *E(Y)*, in contrast to fixed run-length approaches, we generate samples until some error criterion is satisfied.

Two algorithms (SETTLING and CONFIDENCE) are provided to control the convergence. We can monitor the evolution of the various moments AVG for the CONFIDENCE method and AVG, STD, KURT and SKEW for the SETTLING method to precisely quantify the notion of convergence. The Monte Carlo process can be stopped early if these quantities do not vary more than a user-specified limit.

Recall the simulation framework we use when we want to estimate $\theta = E[Y]$:

- we first simulate $Y_1, Y_2, \ldots, Y_n$ and set $\hat{\theta}_n = \dfrac{Y_1 + Y_2 + \ldots + Y_n}{n}$
- the strong Law of Large Numbers says: $\hat{\theta}_n \to \theta$ as $n \to \infty$

But at this point we do not know how large n should be so that we can have confidence in $\hat{\theta}_n$ as an estimator of θ. Put another way, for a fixed value of n, what can be said about the quality of $\hat{\theta}_n$?

One way to answer this question is to use a confidence interval. Suppose we want to estimate θ and we have a random sequence $Y_1, Y_2, \ldots, Y_n$ whose distribution depends on θ. Then we seek $L(Y)$ and $U(Y)$ such that $Prob(L(Y) \le \theta \le U(Y)) = 1 - \alpha$

where $0 \le \alpha \le 1$ is a pre-specified number. We then say that $[L(Y), U(Y)]$ is a $100(1-\alpha)\%$ confidence interval for θ.

There are two types of error that we can consider:

- the absolute error, which is given by $E_a = |\hat{\theta}_n - \theta|$ and
- the relative error, which is given by $E_r = \left|\dfrac{\hat{\theta}_n - \theta}{\theta}\right|$

Now we know that $\hat{\theta}_n \to \theta$ as $n \to \infty$ so that the errors both tend to 0. If $\theta = 0$ then the relative error E_r is not defined. We specify the following error criterion:

Error Criterion: Given $0 \le \alpha \le 1$ and $\varepsilon > 0$, we want $Prob(E \le \varepsilon) = 1 - \alpha$. E is the error type we have specified (relative or absolute).

Suppose that we wish to satisfy the condition $Prob(E_a \le \varepsilon) = 1 - \alpha$, then we continue to generate samples until

$$\frac{\hat{\sigma}_n \cdot z_{1-\alpha/2}}{\sqrt{n}} \le \varepsilon$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$ percentile point of the $N(0, 1)$ distribution and $\hat{\sigma}_n$ is again the estimate of σ based upon the first samples. It is important that n be sufficiently large $n \geq p$, so that $\hat{\theta}_n$ and $\hat{\sigma}_n$ are sufficiently good estimates of θ and σ respectively. As a result, we typically insist that $p = 20$ before we stop.

If we want to control the relative error and have $Prob(E_r \leq \varepsilon) = 1 - \alpha$, then we would simulate samples until

$$\frac{\hat{\sigma}_n \cdot z_{1-\alpha/2}}{\hat{\theta}_n \sqrt{n}} \leq \varepsilon$$

The **SETTLING** method simply controls the absolute and relative errors as if the Monte carlo procedure was a deterministic process. In other words, we use the simple test $E \leq \varepsilon$.

The convergence is associated to a ‘pseudo-extract’ in the Eldo’s language for extraction. The **MCCONV** extract returns 0 or 1 depending on a specific test.

- **MCCONV(**meas_name, **AVG**|**STD**|**KURT**|**SKEW**, **SETTLING**|**CONFIDENCE**[,PARAMETER_LIST]**)**

  Returns 1 or 0 to indicate convergence of the Monte Carlo process for the specified quantity.

  SETTLING

  The Monte Carlo process is stopped if adding new samples no longer changes output by more than a certain threshold. When the **SETTLING** algorithm is specified, the PARAMETER_LIST is: WINDOW[,ABSTOL[,RELTOL].

  CONFIDENCE

  The Monte Carlo is stopped when the confidence interval is small. This algorithm is defined for AVG only; not for STD, SKEW or CURT. When the **CONFIDENCE** algorithm is specified, the PARAMETER_LIST is: WINDOW[,CONFIDENCE_LEVEL[,ABSTOL[,RELTOL]].

Examples:

```
.EXTRACT AC LABEL=gain_db yval(vdb(out), 1MEG)
.EXTRACT MC LABEL=stddev_gain_db_conv
+ MCCONV(gain_db, STD,SETTLING, 0.01, 0.05, 20)
```

The second extract returns a boolean value (1 or 0). It returns 1 when the standard deviation of quantity gain_db has converged to within +/- (0.01 + 5% × M) where M is the average value of the STD taken from the last 20 MC samples.

```
.EXTRACT AC LABEL=gain_db yval(vdb(out), 1MEG)
.EXTRACT AC LABEL=phase_db yval(vp(out), 1MEG)
.EXTRACT DC LABEL=cc i(vdd)
.EXTRACT MC LABEL=mc_std_conv MCCONV(ALL, STD, 20, 0.01,0.05)
```

In this example, the first three **.EXTRACT** statements are considered, and all three of them must have converged before the **MCCONV()** function returns 1.

We recommend to not waste time in tolerance tuning. Use the default values as much as possible, for instance

```
.EXTRACT MC LABEL=mc_avg_conv MCCONV(ALL, AVG, CONFIDENCE)
```

should give a first estimate of the uncertainty, and use the SAVE/RESTART feature to obtain more accurate estimates.

The threshold and sample window size are optional. Their default values can be (re)defined on the **.MC** command.

Extracting the Index and Total Number of Runs

Specify the following arguments to extract the index of the current run and the total number of runs.

- **ICARLO**

  Extracts the index of the current Monte Carlo run (first index is 0)
- **NBCARLO**

  Extracts the total number of Monte Carlo runs

Example:

```
.option aex dump_mcinfo
.param rval=1k dev=10%
v1 0 1 dc 1
r1 1 0 rval
.extract dc label=test {i(v1)/(1+ICARLO)}
.dc
.mc 10 all
```

Statistical Configuration

The keyword, **STATISTICAL**= 0|1, can be specified on X instances, device declarations, or on **.SUBCKT** definitions, to specify whether any statistical variation due to **.MC**, **.WCASE**, or **.DCMISMATCH** can be applied to the specified entities.

If **STATISTICAL** is 0, the selected devices will keep their nominal values. If **STATISTICAL** is 1, the selected devices have statistical variation applied. The global default value can be specified via option **STATISTICAL**= 0|1. Default is 1.

See the Statistical Configuration Usage Example.

Examples

The effects of Monte Carlo analysis can be seen easily in a simple passive low-pass filter.

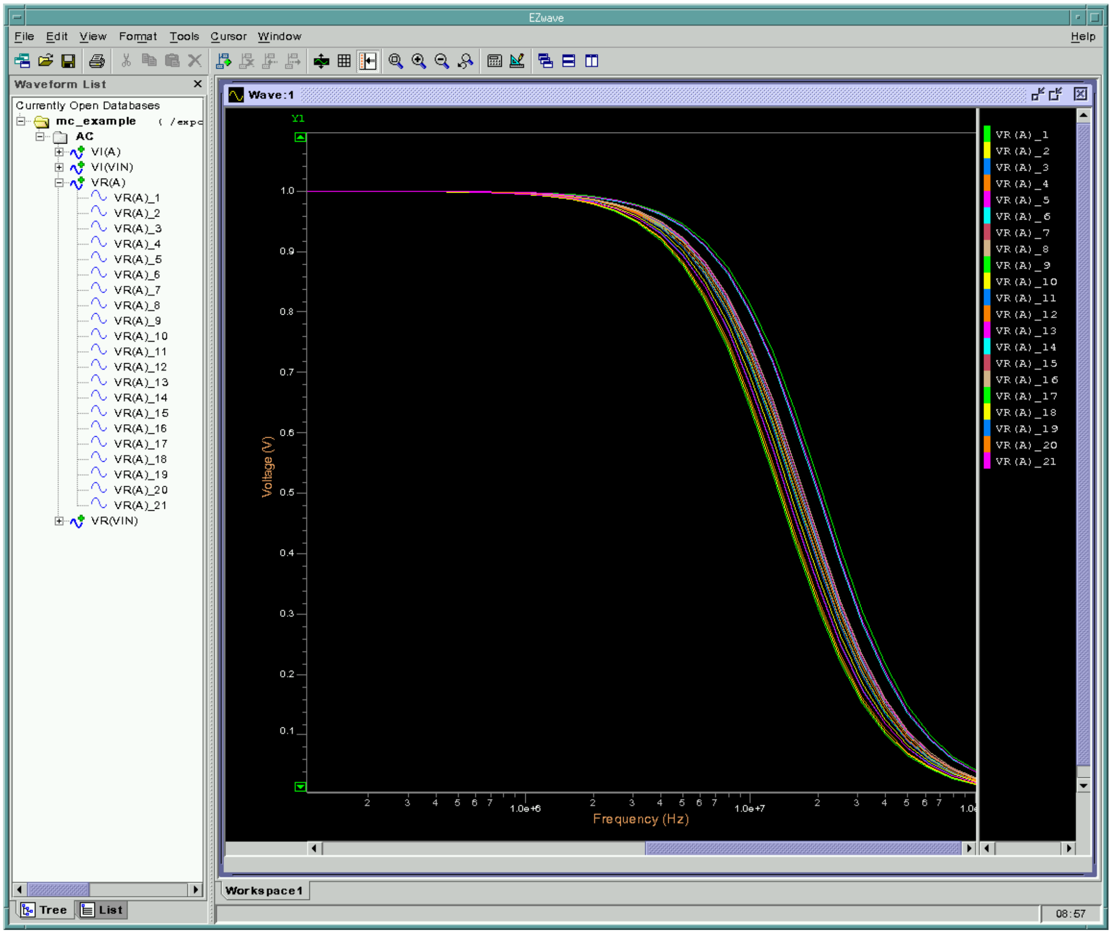
```
.model c cap dev=20%
.model r res lot=20% dev=5%

.mc 20 all

v1 vin 0 ac=1 dc=1
r1 vin a r 1k
c1 a 0 c 10p

.ac dec 10 100 1e8
.probe v
.end
```

The capacitor model has DEV of 20%, so all capacitors using this model are uncorrelated and vary up to 20% from the nominal value specified in the element instantiation. Capacitor `C1` varies uniformly between 8 pF and 12 pF. The resistor model produces resistors that are correlated with a 20% deviation. After Eldo applies the 20% LOT variation, each resistor has a 5% non-correlated variation.

The `ALL` argument to **`.MC`** tells Eldo to save the result from every Monte Carlo iteration. Otherwise, only the average, maximum, and minimum values are saved.

Running this simulation and viewing the voltage at node a produces the following graph.

Figure 19-1. Monte Carlo Analysis Example



Device sizes can be subject to a probabilistic distribution, simulating process variation, through the use of **.PARAM** statements. In this next example, the NMOS transistor's width follows a Gaussian curve, centered around 20e-6 with σ = 0.2×20e-6 = 4e-6 as defined by parameter **nwidth**. This parameter will also be subject to **DEV** variation. A **.EXTRACT** measures the propagation time through the inverter. The *.chi* file contains statistical information about the outcome of the **.EXTRACT**.

```
.model n nmos level=1
+ vto=1.2
+ kp=2.5e-5
+ gamma=1.5

.model p pmos level=1
+ vto=-1.5
+ kp=1.2e-5
+ gamma=1.2

.param nwidth=gauss(20u,0.2,1)
m1 out in vss vss n w=nwidth l=15u
m2 out in vdd vdd p w=30u l=15u
```

```
cout out 0 0.1p

vdd vdd 0 5
vss vss 0 0

vin in 0 pwl (0 0 50n 0 51n 5)

.mc 20 all
.tran 1n 100n
.probe tran v(out)

.extract tran label=tpd tpdud(v(in), v(out))

.end
```

The **.PARAM** usage may be applied to model parameters, as well. The `vto` of the NMOS transistor can be replaced with a Gaussian parameter. In this case, the standard deviation σ is 0.24. This technique may be more useful if the parameter needs to be a mathematical function of several variables.

```
.model n nmos level=1
+ vto=nvto
+ kp=2.5e-5
+ gamma=1.5

.model p pmos level=1
+ vto=-1.5
+ kp=1.2e-5
+ gamma=1.2

.param nvto=gauss(1.2,0.2,1)
m1 out in vss vss n w=20u l=15u
m2 out in vdd vdd p w=30u l=15u

cout out 0 0.1p

vdd vdd 0 5
vss vss 0 0

vin in 0 pwl (0 0 50n 0 51n 5)

.mc 20 all
.tran 1n 100n
.probe tran v(out)

.extract tran label=tpd tpdud(v(in), v(out))

.end
```

Each instance of a transistor using the NMOS model will have the same value of `vto` because `nvto` affects a model parameter and therefore **LOT** variation is used.

The following is an example of the statistical configuration usage:

```
.model r r tc1 = 2 lot = 5% dev = 10%
```

```
.subckt r a b
r1 a b r 1k
.ends

i1 1 0 pwl ( 0 1 10n 10)
x1 1 2 r statistical=0
x2 2 3 r
x2 3 0 r
.tran 1n 10n
.extract dc v(1)
.temp 10
.mc 10
.end
```

In the above, X1.R1 will remain at its nominal value during Monte Carlo analysis. This is because **STATISTICAL** is set to 0 for X1.

```
.model r r tc1 = 2 lot = 5% dev = 10%
.subckt r a b
r1 a b r 1k
.ends

i1 1 0 pwl ( 0 1 10n 10)
x1 1 2 r
x2 2 3 r statistical=1
x2 3 0 r
.tran 1n 10nx2
.extract dc v(1)
.temp 10
.mc 10
.option statistical=0 ! consider devices non-statistical by default
.end
```

In the above, only X2.R1 will vary during Monte Carlo analysis. This is because the global **STATISTICAL** option is set to 0 meaning devices are considered non-statistical by default, but this is overridden by the **STATISTICAL** keyword set to 1 for X2.

The following is an example showing how **SIGBIN** can be specified to truncate the histogram to a certain number of sigmas.

```
* analyze main distribution when there are atypical values
v1 1 0 val
.param val=1 dev/trimodal=0.00002
.distrib trimodal
+ (-1     0) (-0.999 0.1) (-0.998 0)
+ (-0.1   0) ( 0     1  ) ( 0.1   0)
+ ( 0.998 0) ( 0.999 0.1) ( 1     0)
.extract dc label=v1 v(1)
.extract dc label=v1_norm '(v(1)-1.0)*1'
.dc
*
.mc 2000 print_extract=all  sigbin=4
.end
```

Specifying **`SIGBIN`** a value of 4 means atypical values are ignored for computing the min/max for the histogram. Eldo will gather all the samples above “mean+SIGBIN×sigma” in the Above bin, and all the samples below “mean−SIGBIN×sigma” in the Below bin.

Chapter 20
Statistical Experimental Design and Analysis

Introduction

Overview

This chapter is concerned with the standard techniques of factor screening, location and dispersion analysis and response surface methodologies in circuit design. In this context, a general situation is considered where an observable output (extracted measure in Eldo) depends on a set of parameters and the effects (*main effects* and *interactions*) of parameters (*factors*) on the output are explored using the techniques of Design of Experiments.

This release contains only the basic tools to perform a *factor screening* experiment. Because the main issue in studying statistical design problems lies in a large number of circuit parameters involved, the primary purpose of a variable screening experiment is to select or screen out the few *important main effects* from the many less important ones. The Eldo command **`.DEX`** can be used to perform a factor screening. Once a small number of important factors is identified, subsequent statistical design problems can be solved more efficiently and require fewer runs.

In this context, a *factor* is a circuit parameter that is studied in the experiment. In order to study the effect of a factor on one or several user-defined responses, two or more values of the factor are used. These values are referred to as *levels* or *settings*. A combination of factor levels is called a *run* and will be associated to a specific simulation run in Eldo.

In an experiment, we deliberately change one or more variables (or factors) in order to observe the effect the changes have on one or more response variables. The DEX techniques are efficient procedures so that the data obtained can be analyzed to yield valid and objective conclusions. This efficiency is founded on some key properties of orthogonal arrays, which offer a systematic way of testing. The benefits include: a uniform distributed coverage of the test domain, all pair-wise combinations of test set created, and arrive at complex combinations of all the variables.

Practicalities

We wish to emphasize in advance that the procedures implemented in this command should be qualified by the designer. Although many of these methods are part of the current state of the art, the issue of deciding when a computed answer is correct is difficult.

For instance, it is important to note that the quantitative techniques used for factor screening are incomplete in nature because they are numeric summaries. By reducing the data to a few numbers, we filter the data, omit and screen out other sometimes crucial information.

The following lines give some practical considerations:

- The factor screening algorithms use two-level designs, because they require fewer runs and are more economical to conduct. In two-level experiments, only the linear effects can be studied, three-level factors should be considered to detect a curvature effect. For example, the factor "temperature" may affect the responses in a non-monotone way. The use of only two temperature levels ("High" and "Low") represents a strong assumption about the relation between the responses and the temperature.
- The screening designs employed in the DEX analysis are based on Hadamard matrices, which are an important class of orthogonal arrays. In such designs the main effects are, in general, heavily confounded (or aliased) with two-factor interactions. For this reason, it may be difficult to treat the large number of aliased effects and to interpret their significance. It may happen that the main effect analysis does not very well explain the variation in the user-defined response.

Main Effects Analysis and .WCASE Command

When the main effects are investigated, the following main effect model around the nominal point x^0 is assumed:

$$y(x) = y_0 + \sum_{i=1}^{n} \alpha_i (x_i - x^0)$$

As noted above, this model can be applied to the following cases:

- interactions can be tentatively assumed to be zero,
- in screening a large set of factors, some factors with large main effects on the output y exist.

A traditional engineering way to estimate the coefficients a_i has been based on a single or double-sided sensitivity analysis, which can be called the "one-factor-at-a-time" method. This approach is used in the sensitivity computations of the **.WCASE** command. In this case, each of the variables x_i is perturbed by $\pm\Delta x_i$ (where Δx_i is the difference between the "High" and "Low" level values of the *i*th parameter) about the nominal point x^0, keeping all the others at their nominal values.

By comparison with the factorial design method implemented in the DEX procedures, it has the following disadvantages:

- it cannot estimate some interactions,

- the conclusions from its analysis are not general,
- it can miss optimal settings of factors.

Intuitively, this can be seen from the fact that the one-factor-at-a-time plan does not conduct a systematic or comprehensive search over the experimental region.

The **.WCASE** command also provides the so-called worst case values of the response $y(x)$ where x is replaced by the following ‘worst case points’:

$$(x_1 + \Delta x_1, x_2 + \Delta x_2, \ldots, x_n + \Delta x_n), (x_1 - \Delta x_1, x_2 - \Delta x_2, \ldots, x_n - \Delta x_n)$$

where the coordinates are all set to their extreme values: ‘High’ $x_i + \Delta x_i$, or ‘Low’ $x_i - \Delta x_i$.

These points, which are not necessarily the true worst case points, are systematically tested when the selected screening design uses about 2n runs (this is the default argument of the **DESIGN** argument of the **.DEX** command).

Analysis Statement

In order to use the **.DEX** command, the designer must provide the following information:

- the *nominal circuit*, which is identical to the circuit provided for simulation in the netlist format. The user must have a working netlist.
- The *factors*, the designer must specify those parameters that may be considered by the command when conducting the experiment. The designer’s selection of factors is accomplished by adding minor modifications to the working netlist, using the **.PARAMDEX** command. Please refer to the section “.PARAMDEX” on page 1239.
- The *response* may be any quantity represented by a real value, a combination of quantities. This response function is specified by the **.EXTRACT** and **.MEAS** commands.

For example, the **.PARAMDEX** command is used to specify the length `L` and the width `W` of a MOS transistor:

```
.PARAMDEX
+ L CTRL/ REL=20%
+ W CTRL/ REL=20%
```

this command defines the levels (or settings) of L and W as follows:

$$\begin{Bmatrix} L \\ W \end{Bmatrix}_{low} = \begin{Bmatrix} L \\ W \end{Bmatrix}_{nom} - 0.80 \cdot \begin{Bmatrix} L \\ W \end{Bmatrix}_{nom}$$

and:

$$\begin{Bmatrix} L \\ W \end{Bmatrix}_{high} = \begin{Bmatrix} L \\ W \end{Bmatrix}_{nom} + 1.20 \cdot \begin{Bmatrix} L \\ W \end{Bmatrix}_{nom}$$

Running the Experiment

The **`.DEX`** command is part of the Eldo commands. Eldo can be invoked from the command line as follows:

```
eldo <circuit_name> <other_arguments>
```

Commands

The following are commands that can be used for experimental design:

- .DEX
- .PARAMDEX

In the descriptions inside this chapter we use the tokens `IVALUE` and `RVALUE` to denote integer and real (or floating) numbers.

.DEX

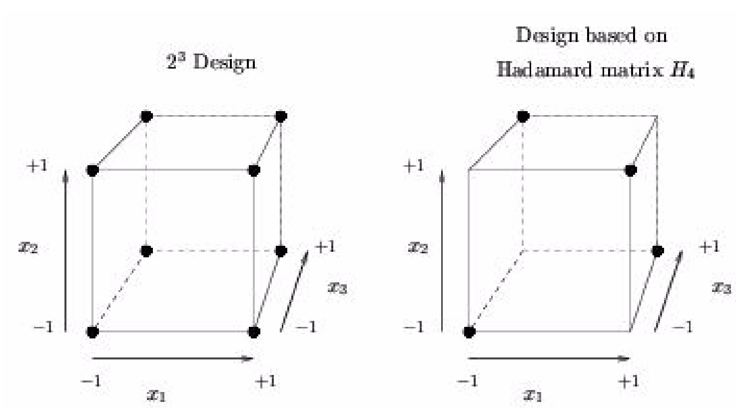
The Eldo command **.DEX** can be used to perform factor screening before attempting to solve subsequent statistical problems. The primary purpose of a variable screening experiment is to select or screen out the few important main effects from the many less important ones. Note that the **.DEX** command is compatible with the Multiple Run features (see the command description for ".MPRUN" on page 729).

The maximum number of factors N_{max} is 40000. This number is the result of the algorithm we used for the construction of Hadamard matrices. All Hadamard matrices of orders N up through 28, and at least one of every order N up through 200 are not computed, but stored explicitly in the static memory. Higher orders are obtained implicitly by computing Kronecker products $H_1 \otimes H_2$. If H_1 and H_2 are Hadamard matrices of order N_1 and N_2, then the previous direct product is a Hadamard matrix of order $N_1 N_2$.

It is not recommended to run very large experiments. Most of our test-cases involved a moderate number of factors, say up to 50. Note that the range 7 to 150 should be considered as a safe area of operatability for the Lenth's method.

Figure 20-1 depicts the geometrical differences between a full factorial design and an orthogonal arrays based on Hadamard matrices.

Figure 20-1. Cuboidal representation of two 3-factors designs



The black bullets indicate that the *corners* are effectively chosen. The first design is obtained by setting the argument **DESIGN=FULL_FACT** and the second by **DESIGN=ORTHA_2_N**. One can clearly observe that the last design is a balanced subset of the full factorial design.

Command Syntax

The main effect analysis acting on all the circuit parameters defined with the **.PARAMDEX** statement is defined as follows:

```
.DEX
+ EXPERIMENT = SCREENING | SCREENING_CTRL | SCREENING_NOISE
```

```
+ RESPONSE = LIST_OF_MEASURES
+ [DESIGN = ORTHA_2_N | ORTHA_2_2N | FULL_FACT]
+ [FACTOR = LIST_OF_FACTORS]
+ [FIND_FACTOR]
```

Parameters

- **EXPERIMENT**=**SCREENING**|**SCREENING_CTRL**|**SCREENING_NOISE**

  A screening experiment will be conducted on the performance response defined with the argument **RESPONSE**. The factors will be taken from the list of circuit parameters appearing on a **.PARAMDEX** statement. The results of this experiment are available in the *.dex* file. This output file contains an ordered data table for each simulated response, a DEX mean table which provides the ranked list of factors (not including the interactions), the ranking is from the most important factor to least important factor, and a formal test of significance based on the Lenth's method. The additional analyses **SCREENING_CTRL** and **SCREENING_NOISE** can be used to filter out the noise and designable parameters respectively for a screening experiment. This option is particularly useful for identifying the critical noise factors and critical designable factors separately, please consult the section "Two-stage Strategy for the Identification of Critical Factors" for details of this approach.

- **RESPONSE**=LIST_OF_MEASURES

  List of comma-separated measures to be considered in the experiment.

- **FACTOR**=LIST_OF_FACTORS

  List of comma-separated circuit parameters specified with the **.PARAMDEX** command to screen out. By default, all the circuit parameters defined with the **.PARAMDEX** command are used in the factor screening.

- **DESIGN**=ORTHA_2_N|ORTHA_2_2N|FULL_FACT

  The category of the design matrix used to run the experiment. The arguments **ORTHA_2_N** and **ORTHA_2_2N** specify that an orthogonal array at two levels with economical run size based on Hadamard matrices has to be chosen. If N is the number of factors, the design **ORTHA_2_N** will use about N runs, and **ORTHA_2_2N** and $2N$ runs. A full factorial design 2^N can also be used when the number of factors is small (say $N < 7$), because it requires 2^N runs.

  Default is: **ORTHA_2_2N**.

- **FIND_FACTOR**

  This option will return the list of process parameters (noise factors) which are candidates for the **.DEX** command. When this argument is activated the DEX analysis is not performed.

Results and Output of the SCREENING Experiment

The results of this analysis are presented below. The example used is *cmos_ring.cir* (available in the directory: *$MGC_AMS_HOME/examples/optimizer/*). Two response functions are defined which are the oscillation frequency 30 MHz < `OSCFREQ` < 50 MHz, and the AC switching power dissipation: `ACPOWER` < 5 MW.

This analysis is performed with respect to 34 noise factors. Note that the *.dex* file gives the name and the properties of each factor. A dummy name is associated to each factor in order to get more compact tables. The control factors are designated by `C1`, `C2`, `C3`, ... and the noise factors by `N1`, `N2`, `N3`, and so on.

```
Factors and Levels
 ------------------

 ==> Level values and coded factors

 ==> Output in the following order:
     Distrib   ... Statistical distribution of factor (Uniform or Gaussian)
      Coding    ... Levels are coded with the Std. Deviation
      Mult      ... Fraction of the Std. Deviation or the Range
      Dev/Range ... Std. Deviation or Range
      Mean      ... Mean value of the statistical distribution
      Low/High  ... Low and High settings (coded with -1 and +1)


Noise Factor(s)         Distrib  Coding  +/-Mult     Dev/Range
 N1 - M(SBSIMP,X3U1)    Gauss    StdDev  1.0e+00    2.8900e-03 ...
 N2 - M(SBSIMP,X3MS)    Gauss    StdDev  1.0e+00    1.4400e+00 ...
 N3 - M(SBSIMP,X2MS)    Gauss    StdDev  1.0e+00    1.4700e+00 ...
 N4 - M(SBSIMP,MUS)     Gauss    StdDev  1.0e+00    2.2100e+01 ...
 N5 - M(SBSIMP,X2U1)    Gauss    StdDev  1.0e+00    2.1400e-03 ...
 N6 - M(SBSIMP,X2U0)    Gauss    StdDev  1.0e+00    7.7600e-04 ...
 N7 - M(SBSIMP,X3E)     Gauss    StdDev  1.0e+00    7.6500e-04 ...
 N8 - M(SBSIMP,X2E)     Gauss    StdDev  1.0e+00    8.2600e-04 ...
 N9 - M(SBSIMP,X2MZ)    Gauss    StdDev  1.0e+00    7.2600e-01 ...

...
 N29 - M(SBSIMN,MUZ)     Gauss    StdDev  1.0e+00    4.6100e+01 ...
 N30 - M(SBSIMN,ETA)     Gauss    StdDev  1.0e+00    5.3200e-03 ...
 N31 - M(SBSIMN,K2)      Gauss    StdDev  1.0e+00    3.7800e-02 ...
 N32 - M(SBSIMN,K1)      Gauss    StdDev  1.0e+00    1.1900e-01 ...
 N33 - M(SBSIMN,PHI)     Gauss    StdDev  1.0e+00    1.0800e-02 ...
 N34 - M(SBSIMN,VFB)     Gauss    StdDev  1.0e+00    8.0600e-02 ...
```

Ordered Data Table

An ordered data table for each simulated response, this data table answers the following basic question: what is the best setting (based on the data) for each of the N factors? An ordered data table is formed by:

- vertical axis: the ordered (smallest to largest) raw response value for each runs in the experiment.
- horizontal axis: the corresponding factor index with (at each run) a designation of the corresponding settings (-1 or +1) for each of the N factors.

This analysis can be found in the *.dex* file as follows:

```
Ordered Data Table
```

```
------------------

Response Y : OSCFREQ

Mean Value             :  4.8667e+07
Value at nominal point :  4.8667e+07

==> Vertical axis: the ordered raw response
    value for each of the 72 runs in the experiment.

    Horizontal Axis: the corresponding dummy variable index (1 to 34)
    with settings (-1 or +1) for each of the 34 factors.


Run  N1   N2   N3   N4    ...  N29  N30  N31  N32  N33  N34     Response Y
  4  +1   -1   -1   -1         +1   -1   +1   +1   +1   -1    4.1226377e+07
 56  -1   +1   +1   -1         +1   -1   +1   +1   +1   +1    4.1369450e+07
 31  -1   -1   +1   -1         -1   -1   -1   +1   +1   +1    4.2162628e+07
 72  +1   +1   -1   -1         -1   -1   -1   -1   -1   +1    4.2545442e+07
 15  -1   -1   +1   -1         -1   +1   +1   -1   +1   +1    4.2712903e+07
 61  +1   +1   -1   +1         +1   +1   -1   +1   -1   +1    4.3002165e+07
 11  +1   -1   +1   -1         +1   +1   -1   +1   +1   -1    4.3099024e+07
 14  -1   +1   -1   +1         +1   +1   -1   +1   +1   -1    4.3174847e+07
 19  -1   -1   +1   -1    ...  +1   -1   -1   -1   -1   +1    4.3490089e+07
 ...
```

DEX Mean Table

A DEX mean table which provides the ranked list of factors (not including the interactions), the ranking is from the most important factor to least important factor. This table is appropriate for analyzing data from an experiment, with respect to important factors, where the factors are at two levels. The table gives the mean value and the amplitude of the main effect coefficient (a line proportional to this magnitude is also printed). Note that the magnitude represents the absolute value of the coefficient a_i.

This qualitative analysis can be found in the *.dex* file as follows (the central part has been removed):

```
DEX Mean Table
 --------------

 Response Y : OSCFREQ

 ==> Horizontal axis: the mean response for a given setting (- or +)
     of a factor, for each of the 34 factors.

     Vertical axis: the 34 factors and the two settings (- and +)
     within each factor.


   Value at nominal point :  4.8667e+07
   Mean Value             :  4.8667e+07

 Factor       Name            Sign of            Magnitude
```

```
                                    effect   (percent / exact / cumulative)
     N4          M(SBSIMP,MUS)       [+]      13.0%   3.6552e+06    13.0%
     N27         M(SBSIMN,U1)        [-]      12.8%   3.6022e+06    25.8%
     N21         M(SBSIMN,MUS)       [+]      11.6%   3.2592e+06    37.3%
     N32         M(SBSIMN,K1)        [-]       9.9%   2.7839e+06    47.2%
     N15         M(SBSIMP,K1)        [-]       7.8%   2.1921e+06    55.0%
     N10         M(SBSIMP,U1)        [-]       7.5%   2.1091e+06    62.5%
     N17         M(SBSIMP,VFB)       [-]       6.0%   1.6999e+06    68.5%
     N34         M(SBSIMN,VFB)       [-]       5.8%   1.6240e+06    74.3%
     N18         M(SBSIMN,X3U1)      [+]       3.1%   8.7887e+05    77.4%
     N11         M(SBSIMP,U0)        [-]       2.8%   7.8008e+05    80.2%
     ...
     N23         M(SBSIMN,X2U0)      [-]       0.0%   1.5643e+03   100.0%
     N9          M(SBSIMP,X2MZ)      [-]       0.0%   7.2275e+02   100.0%
```

The last column gives the following information: it says that almost 70% of the response's variability is explained with the variations of the eight factors `M(SBSIM*,MUS)`, `M(SBSIM*,U1)`, `M(SBSIM*,K1)` and `M(SBSIM*,VFB)` where the '`*`' character means '`P`' positive or '`N`" for negative.

Formal Test of Significance

A formal test of significance is based on the Lenth's method. A detailed description of this algorithm can be found in [1]. This method is usually employed for unreplicated experiments, where there are no replicated runs and no dispersion modeling is investigated.

This quantitative analysis is given as follows:

```
   ==> Response Y[1] : OSCFREQ

    ==> Initial standard error s0      :  5.8724e+05
        Pseudo standard deviation PSE :  3.6619e+05
        Signifiance level alpha       : 0.05
        Critical value                :  2.0525e+00

     Main        Name              Accept/Reject      Value t(PSE)      P-value
    Effect

    N4          M(SBSIMP,MUS)        *** PASS ***        9.9816e+00        0.001
    N27         M(SBSIMN,U1)         *** PASS ***        9.8368e+00        0.001
    N21         M(SBSIMN,MUS)        *** PASS ***        8.9002e+00        0.001
    N32         M(SBSIMN,K1)         *** PASS ***        7.6022e+00        0.001
    N15         M(SBSIMP,K1)         *** PASS ***        5.9861e+00        0.001
    N10         M(SBSIMP,U1)         *** PASS ***        5.7594e+00        0.001
    N17         M(SBSIMP,VFB)        *** PASS ***        4.6422e+00        0.001
    N34         M(SBSIMN,VFB)        *** PASS ***        4.4348e+00        0.002
    N18         M(SBSIMN,X3U1)       *** PASS ***        2.4000e+00        0.03
    N11         M(SBSIMP,U0)         *** PASS ***        2.1302e+00        0.05
    N28         M(SBSIMN,U0)                             1.7737e+00        0.09
    N31         M(SBSIMN,K2)                             1.7069e+00        0.1
    N29         M(SBSIMN,MUZ)                            1.5284e+00        0.2
    N1          M(SBSIMP,X3U1)                           1.5215e+00        0.2
    N12         M(SBSIMP,MUZ)                            1.4868e+00        0.2
    N16         M(SBSIMP,PHI)                            1.3491e+00        0.2
   ...
```

Non-expert users have only to consider the central column `Accept/Reject`. Based on our experience, the Lenth's method is not very restrictive. Inactive factors can pass the test. We suggest that misidentifying an inert factor as active should be less important than missing an important and active factor.

Multi-response Problems

In the case of multi-response problems, a summary of the screening experiments is performed for each response.

```
Active effect(s) for 2 response(s) :
 N4    -   M(SBSIMP,MUS)   Rank = 33      Response(s) :   OSCFREQ   ACPOWER
 N27   -   M(SBSIMN,U1)    Rank = 32      Response(s) :   OSCFREQ   ACPOWER
 N21   -   M(SBSIMN,MUS)   Rank = 31      Response(s) :   OSCFREQ   ACPOWER
 N32   -   M(SBSIMN,K1)    Rank = 30      Response(s) :   OSCFREQ   ACPOWER
 N15   -   M(SBSIMP,K1)    Rank = 29      Response(s) :   OSCFREQ   ACPOWER
 N10   -   M(SBSIMP,U1)    Rank = 28      Response(s) :   OSCFREQ   ACPOWER
 N17   -   M(SBSIMP,VFB)   Rank = 27      Response(s) :   OSCFREQ   ACPOWER
 N34   -   M(SBSIMN,VFB)   Rank = 26      Response(s) :   OSCFREQ   ACPOWER
 N18   -   M(SBSIMN,X3U1)  Rank = 25      Response(s) :   OSCFREQ   ACPOWER
 N11   -   M(SBSIMP,U0)    Rank = 24      Response(s) :   OSCFREQ   ACPOWER


 Active effect(s) for 1 response(s) :None


 Unimportant effect(s) detected for all responses:

 N9      -   M(SBSIMP,X2MZ)
 N33     -   M(SBSIMN,PHI)
 N12     -   M(SBSIMP,MUZ)
 N13     -   M(SBSIMP,ETA)
 N8      -   M(SBSIMP,X2E)
 N3      -   M(SBSIMP,X2MS)
 ...
```

This global analysis based on the Lenth's method clearly identifies the eight factors `M(SBSIM*,MUS)`, `M(SBSIM*,U1)`, `M(SBSIM*,K1)`, `M(SBSIM*,VFB)`, and the additional `M(SBSIMN,X3U1)`, `M(SBSIMP,U0)` as active factors.

Unordered Data Table (CSV Table)

An unordered data table gives the simulation results. These results are stored in a table with the CSV format (Comma Separated Values). CSV is a delimited data format with fields/columns separated by the comma character and records/rows separated by new lines.

Results of the FIND_FACTOR Process

The **`FIND_FACTOR`** option can be used to determine the list of noise parameters prior to any DEX analysis. It will be used to prepare the **`.PARAMDEX`** statements. This information can be found into two different output files: the *.dex* file contains formatted data, and the include file *_paramdex.inc* file which contains the related **.PARAMDEX** command.

Consider the following netlist (from the example *filter.cir*):

```
* CIRCUIT
V1 1 0 DC 0 AC 1
R1 1 2     'R1 + D_R1'
R3 2 0     'R3 + D_R3'
C2 2 INN   'C + D_C'
C3 2 OUT   'C + D_C'
R2 OUT INN 'R2 + D_R2'
Y1 OPAMP1 0 INN OUT 0 PARAM: GAIN='GAIN + D_GAIN' P1='P1 + D_P1'

* DETERMINISTIC PARAMETERS
.PARAM R1 = 159K R2 = 3.18MEG R3 = 79.5
+ C = 1U
+ P1 = 10 GAIN = 10E6

* NOISE PARAMETERS
.PARAM D_R1 = AGAUSS(    0,    10.6K,  1)
.PARAM D_R2 = AGAUSS(    0, 0.212MEG,  1)
.PARAM D_R3 = AGAUSS(    0,     5.30,  1)
.PARAM D_C = AGAUSS(    0U,   0.067U,  1)
.PARAM D_P1 = AGAUSS(    0,   0.667U,  1)
.PARAM D_GAIN = AGAUSS(  0,  0.667E6,  1)
```

By running Eldo with the following **.DEX** command

```
.DEX
+ EXPERIMENT=SCREENING_NOISE
+ DESIGN=ORTHA_2_N
+ RESPONSE=OPGAIN,OPFREQ,OPBW
+ FIND_FACTOR
```

the *.dex* file will contain the list of noise parameters as follows:

```
List of candidates for the DEX analysis
 ---------------------------------------

    Noise Factor(s)        Dummy    Distrib      Nominal      Dev/Range
                           Name                   Value

D_GAIN            SIG=1       N1      Gaussian    0.0000e+00    6.6700e+05
D_P1              SIG=1       N2      Gaussian    0.0000e+00    6.6700e-07
D_C               SIG=1       N3      Gaussian    0.0000e+00    6.7000e-08
D_R3              SIG=1       N4      Gaussian    0.0000e+00    5.3000e+00
D_R2              SIG=1       N5      Gaussian    0.0000e+00    2.1200e+05
D_R1              SIG=1       N6      Gaussian    0.0000e+00    1.0600e+04
```

The following **.PARAMDEX** command will be found in the file *filter_paramdex.inc*:

```
.PARAMDEX
+   D_GAIN   NOISE/SIG=1.0
+   D_P1     NOISE/SIG=1.0
```

```
+   D_C     NOISE/SIG=1.0
+   D_R3    NOISE/SIG=1.0
+   D_R2    NOISE/SIG=1.0
+   D_R1    NOISE/SIG=1.0
```

The .dex File and the viewdex Script

The *.dex* file is a structured text file that is organized with sections and sub-sections, and allows the user to skip one or more sections at different levels of detail. The user has to run the service routine named `viewdex` on the *.dex* file in order to display the results. This routine can be invoked as follows:

```
viewdex [-l d] -f file.dex
```

with the arguments:

- `-l d`

  Where `d` is the level of detail that users wants to view. The range of levels is 1 to 3, level 1 (default) gives the least amount of data while level 3 gives the most amount of data.

  - Level 1: only the results of the screening experiment are printed for multi and simple response problem.
  - Level 2: in the case of multi-responses problem, the results of screening experiments for each response are given separately.
  - Level 3: the script gives the ordered and unordered data tables (in CSV format) and the design matrix.

When an outer loop has been performed, the amount of data can be very large. In this case level 1 only reports simplified results in a table printed at the end of the *.dex* file.

- `-f`

  Name of the *.dex* file to be formatted.

.PARAMDEX

The formulation of practical problems requires the consideration of two different types of circuit parameters. To this purpose, the following classification of parameters is proposed:

- deterministic or *designable* parameters x, we will also use the term *control parameters*;
- statistical or hard to control parameters s, we will use the term *noise parameters*. Usually these parameters are defined in one or several **.MODEL** statements with the arguments **DEV**, **DEVX** and **LOT**. The levels for these factors are defined as a fraction of the standard deviation (or range) of their probability distribution.

Integrated circuit design features all two types of parameters. The transistor geometries are deterministic design parameters. The transistor model parameters are statistical parameters that reflect inevitable manufacturing fluctuations. On the other hand, in discrete RLC circuits the designable parameters and the statistical variables are in the same space. These notions are explained in details in section "Notes on Statistical Modeling for Discrete Circuits".

When writing the **.PARAMDEX** statements, the user has to choose the range of the settings for the input factors, and it is wise to give this some thought beforehand rather than just try extreme values. In some cases, extreme values will give runs that are not feasible. Note that these runs are simply ignored during the post-analysis of the experiment.

The actual **.DEX** command allows for two-level designs that have just "High" and "Low" settings for each factor. The most popular experimental designs are two-level designs. One reason why two is the most common choice is that it is ideal for screening designs, simple and economical; it also gives most of the information required to go to a multilevel response surface experiment if one is needed.

Command Syntax

The noise factors can be specified with the **.PARAMDEX** commands:

```
.PARAMDEX FACTOR_NAME [NOISE/] SIG=NSIGMA
.PARAMDEX FACTOR_NAME [NOISE/] RNG=NRANGE
.PARAMDEX FACTOR_NAME [NOISE/] ABS=DELTA
.PARAMDEX FACTOR_NAME [NOISE/] REL=DELTA%
```

and the designable factors with the following commands:

```
.PARAMDEX FACTOR_NAME [CTRL/] ABS=DELTA [,NOM=RVALUE]
.PARAMDEX FACTOR_NAME [CTRL/] REL=DELTA% [,NOM=RVALUE]
```

Parameters

- FACTOR_NAME

  Name or a reference to a factorial parameter. This parameter can be one of the following:

```
PARAMETER_NAME |
P(PARAMETER_NAME) |
P(SUBCKT_NAME,SUBCKT_PARAMETER_NAME) |
E(DEVICE_NAME,PARAMETER_NAME) |
M(MODEL_NAME,MODEL_PARAMETER_NAME) |
EM(DEVICE_NAME,MODEL_PARAMETER_NAME)
```

These constructs are strongly related to the statical model the user want to define, please consult the section "Selection of Factors for the DEX Analysis" for details. Note that the **P**(.,.) construct is not fully implemented in release 2007.1.

- NOISE, CTRL

  This optional argument specifies the category of the factor. Eldo will automatically determine this category, but in order to clarify the netlist it is recommended to write **.PARAMDEX** commands with these flags.

- SIG=NSIGMA

  Fraction of the standard deviation used to define the levels of the factor. The high and low levels are defined with $\pm n \cdot \sigma$ where σ is the standard deviation defined on the parameter (see section "Normal and Uniform Distributions" for details).

- RNG=NRANGE

  Fraction of the half range used to define the levels of the factor associated to a uniform distribution. The high and low levels are defined with $\pm n \cdot h$ where h is the half range defined on the parameter (see section "Normal and Uniform Distributions" for details).

- **ABS**=DELTA

  The levels are specified with respect to the nominal value x_{nom} as an absolute perturbation of size $\pm\Delta$.

- **REL**=DELTA%

  The levels are specified with respect to the nominal value x_{nom} as a relative perturbation of size $\pm\Delta \cdot x_{nom}$ (see section "Levels are specified with respect to the nominal value" for details of **REL** and **ABS** arguments).

- **NOM**=RVALUE

  This argument allows to redefine the nominal value of a designable factor. It is not possible to modify the nominal value of noise factors, since it is part of the user-defined statistical model.

Two-stage Strategy for the Identification of Critical Factors

A screening experiment is performed to identify the important factors. It is usually referred to as *phase zero* of a response surface study. The following lines give some guidelines on how screening designs may be conducted.

In the definition of the **.PARAMDEX** command we began by emphasizing the importance of the control/noise distinction in the statement of IC design. The regression model used in the DEX analysis is defined as follows:

$$y(x, s) = y_0 + \sum_{i=1}^{Nc} \alpha_i(x_i - x^0) + \sum_{i=1}^{Nn} \beta_i(s_i - s^0)$$

The y_0, α ‘s and the β ‘s are a set of unknown coefficients. To estimate the values of these parameters, the DEX analysis collect data on the system we are studying. When no distinction is made among the factors, the y_0, α, β are estimated within the same experiment. This kind of experiment is only realized when **EXPERIMENT**=**SCREENING**.

We suppose that a two-stage strategy could be used in the case of integrated circuit design, where the designable and noise variables are defined in separate spaces. It may happen that some noise factors, such as environmental noise factors, have large masking effects. The phase zero will then involve two steps:

- a screening experiment for noise factors should be performed first by not considering the designable factors with **EXPERIMENT**=**SCREENING_NOISE**. The DEX analysis run the experiment by using $x = x^0$, where x^0 is the experiment center in the space of control variables to get the amplitude of the β ‘s;
- by setting **EXPERIMENT**=**SCREENING_CTRL** a screening for the control factors would then be launched. The amplitude of the α ‘s are then analyzed to reduce the list of candidate variables to a relatively few. In this case the noise variables are fixed to their mean value s^0.

Noise Factors with DEV and LOT Variations

In statistical models for transistor level simulation, there are two types of variability in device properties, often referred to as *inter-die* (between chips) and *intra-die* (within chip) variations. Inter-die variability consists of the accumulated fluctuations of material characteristics form lot-to-lot, wafer-to-wafer and die-to-die (chip-to-chip). However, once the wafers have been cut into individual chips there is no longer a traceable correlation among them. Therefore, lot-to-lot, wafer-to-wafer and die-to-die variations are pooled together into one term: the inter-die variability. This variability equally affects all devices on a given chip. In simulation terms, this means that all devices of a certain type use the same statistical model.

The Eldo simulator allows the specification of two different device-to-device variations. For each run of the sampling algorithm (for instance the Monte Carlo method), the parameters with **DEV** variations receive a random value each time their symbol is used in expressions. Suppose the following statements:

```
.PARAM param_stat=value DEV=value

.PARAM param1=function_of(param_stat)
.PARAM param2=function_of(param_stat)

.PARAM param3=function_of(param1,param2)
```

The sampling process will compute one value for the parameter `param_stat` to get the value of `param1` and a different random value for `param2` (according to the underlying probability distribution defined on `param_stat`). Therefore, the expression `param3` is a function of two independent statistical parameters, say `param1_stat` and `param2_stat`, which share the same probability distribution. In the DEX analysis, a *unique factor* is defined. This feature comes from two facts:

- the Eldo's extraction language does not allow to make explicit reference to the parameters `param1_stat` and `param2_stat`,
- it is always possible to rewrite the expression of `param3` as a function of `param_stat`:

```
.PARAM param_stat=value DEV=value

.PARAM param3=function_of(param_stat)
```

It means that the sensitivity of the function `param3` with respect to the formal parameter `param_stat` is *not* investigated with the parameters `param1_stat` and `param2_stat2` independently.

Selection of Factors for the DEX Analysis

This section describes how the circuit parameters (the noise and control factors) have to be referenced in the **.PARAMDEX** statements. We will consider different situations, but we did not explore all the possibilities. These notes came from our experiments.

We define three statistical parameters, each one has different statistical variations. The global parameter `param4_global` has no statistical variations.

```
* PARAMETERS DEFINITION
.PARAM param1_global=value DEV=value
+      param2_global=value DEVX=value
+      param3_global=value LOT=value

.PARAM param4_global=value
```

A model `MOD1` is defined with two parameters as follows,

```
* MODEL DEFINITION
.MODEL MOD1 ...
+ param1_model=value DEV=value param2_model=value LOT=value ...
```

and a device named `MA`,

```
MA ... MOD1 param1_inst=value ... param1_model=value param2_model=value ...
```

We also assumed that the model `MOD1` has some instance parameters named `param1_inst`, and so on.

Suppose that a screening design has to be performed on our formal circuit. The **.PARAMDEX** statements associated to the model parameters `param1_model` and `param2_model` are necessary of the form:

```
.PARAMDEX
+ EM(MA,param1_model) [NOISE/] ...
+ M(MOD1,param2_model) [NOISE/] ...
```

The construct **EM**(.,.) always selects a model parameter associated to a specific instance, hence it means that this parameter *must* have DEV or DEVX variations. The second construct **M**(.,.) will always be associated to a parameter with LOT variations, since it refers globally to a model parameter.

Based on the previous example, we can add a **.SUBCKT** definition.

```
* SUBCIRCUIT DEFINITION
.SUBCKT CKT1 ... PARAM: param1_ckt=value
.PARAM param1_loc=value
M1 ... MOD1 param1_inst=value ... param1_model=value ...
M2 ... MOD1 param1_inst=value ... param1_model=value ...
...
R1 ... value
...
.ENDS CKT1

* CIRCUIT
MA ... MOD1 param1_inst=value ... param1_model=value param2_model=value ...
MB ... MOD1 param1_inst=value ... param1_model=value param2_model=value ...

XA ... CKT1 PARAM: param1_ckt=value
XB ... CKT1 PARAM: param1_ckt=value

RA ... function_of(param1_global,...)
```

Suppose we wish to perform a screening experiment on some instances `M1`, `M2`, ... with respect to the parameter `param1_model` and the resistor devices `R1` and `RA`. The following statements

```
.PARAMDEX
+ EM(XA.M1,param1_model) [NOISE/] ...
+ EM(XA.M2,param1_model) [NOISE/] ...
...
+ EM(XB.M1,param1_model) [NOISE/] ...
```

will select the factors `param1_model` associated to the instances `XA.M1`, `XA.M2`, ..., `XB.M1`, and so on.

Both the following statements will select the resistor value of the `X1.R1` and `RA` devices.

```
.PARAMDEX
+ E(XA.R1,R) [CTRL/] ...
...
+ E(RA,R) [CTRL/] ...
```

Note that capacitance and inductance values can also be defined with the selectors `E(.,C)` and `E(.,L)` respectively. More generally, instance parameters, such as lengths and widths of transistors, *must* be also selected with the `E(.,.)` construct. For instance, if `MOD1` is a transistor model, the `XA.M1` device will have instance parameters `W`, `L`, `AD`, `AS`, and so on. These parameters will be selected with `E(XA.M1,W)`, `E(XA.M1,L)`, ... The general syntax is defined as follows:

```
.PARAMDEX
+ E(XA.M1,param1_inst) [CTRL/] ...
...
+ E(XA.M2,param1_inst) [CTRL/] ...
...
```

The last selector `P(.,.)` allows to manipulate subcircuit parameters. This feature is not completely available in the release 2007.1 but we will describe briefly how it is specified to complete these notes.

We are considering this situation:

```
.PARAM param4_global=value

* SUBCIRCUIT DEFINITION
.SUBCKT CKT1 ... PARAM: param1_ckt=value
.PARAM param1_loc=value
M1 ... MOD1 param1_inst=function_of(param4_global) ... param1_model=value ...
M2 ... MOD1 param1_inst=value ... param1_model=value ...
...
R1 ... value
...
.ENDS CKT1
```

The global parameter `param4_global` has no random variations. It is therefore a deterministic parameter and can only be selected with these equivalent statements:

```
.PARAMDEX param4_global [CTRL/] ...

.PARAMDEX P(param4_global) [CTRL/] ...
```

The next cases are represented by the explicit and implicit circuit parameters. The parameters `param1_ckt` and `param1_loc` can be selected with the `P(.,.)` selector.

```
.PARAMDEX P(XA,param1_loc) [CTRL/] ...

.PARAMDEX P(XA,param1_ckt) [CTRL/] ...
```

Note that the implicit definition of the local parameter `param1_loc` is not native for the Eldo's language, and should be avoided by the user as much as possible. An explicit definition offers a better understanding.

Now suppose we want to select the factors associated to the global parameters `param1_global` and `param2_global`.

```
* SUBCIRCUIT DEFINITION
```

```
.SUBCKT CKT1 ... PARAM: param1_ckt=value
M1 ... MOD1 param1_inst=function_of(param1_global,param2_global,...) ...
M2 ... MOD1 param1_inst=function_of(param1_global,param2_global,...) ...
...
R1 ... value
...
.ENDS CKT1
```

For the parameter `param1_global` with DEV variations, this list of factor can be large:

```
.PARAMDEX
+ P(XA.M1,param1_global) [NOISE/] ...
+ P(XA.M2,param1,global) [NOISE/] ...
...
+ P(XB.M1,param1_global) [NOISE/] ...
```

On the other hand, for DEVX variations, the only allowed construct is:

```
.PARAMDEX
+ P(XA,param1_global) [NOISE/] ...
```

since there is only one ‘private’ parameter attached to each sub-circuit instance.

Examples

These examples are extracted from the testcase *cmos_delay.cir* (available in the directory: *$MGC_AMS_HOME/examples/optimizer/*). This circuit contains three groups of parameters:

- the process parameters used in a Level 8 NMOS model, such as the Oxide Thickness `TOX`, NMOS Length and Width Reductions (`DLN`, `DWN`), ...
- a second set of environmental noise parameters such as two supply voltages (`VPERI`, `VBB`) and the temperature `TP`,
- and the designable variables, the length and width of transistors

Suppose our approach is to assume that process disturbances affect devices within the same chip in the same way: only inter-die variations exist and intra-die variations are negligible. The set of process parameters is common to all NMOS and PMOS devices within the same chip. These statistical definitions can be translated as follows:

```
.PARAM TOX   = AGAUSS(    0.015,  0.0003,  1)
+      DL    = AGAUSS(    0.05,  0.0044,  1)
+      DW    = AGAUSS(    1.00,  0.044,   1)
+      VFB   = AGAUSS( -0.8632,  0.0186,  1)
```

Then the associated **.PARAMDEX** statements could be defined as follows:

```
.PARAMDEX
+ DL    NOISE/SIG=2.5
+ DW    NOISE/SIG=2.5
+ TOX   NOISE/SIG=2.5
+ VFB   NOISE/SIG=2.5
```

Note that these statements are equivalent to the following:

```
.PARAMDEX
+ P(DL)     NOISE/SIG=2.5
+ P(DW)     NOISE/SIG=2.5
+ P(VFB)    NOISE/SIG=2.5
+ P(TOX)    NOISE/SIG=2.5
```

The next examples illustrate the use of the constructs `M(.,.)` and `EM(.,.)` with the inter and intra variations. The use of `M(.,)` is not very useful in this case but we will assume that the statistical variations were stated as follows:

```
.PARAM TOX=0.015
.MCMOD NCH TOX LOT/GAUSS=0.0003
.MCMOD PCH TOX LOT/GAUSS=0.0003
```

There are two independent random variables `TOX` associated to the PMOS and NMOS transistor. Two PMOS, or respectively two NMOS, transistors are assumed to be perfectly tracking each other, but PMOS and NMOS transistor have independent variations. This is a very hypothetical situation, since the Oxide Thickness should affect the PMOS and NMOS transistors in the same way. The associated **`.PARAMDEX`** command would be as follows,

```
.PARAMDEX
+   M(PCH,TOX)  SIG=2.5
+   M(NCH,TOX)  SIG=2.5
```

Suppose now we are interested in intra-die variations. The statistical variations can be defined as follows:

```
.PARAM TOX=0.015
.MCMOD NCH TOX DEV/GAUSS=0.0003
.MCMOD PCH TOX DEV/GAUSS=0.0003
```

It follows that the associated **`.PARAMDEX`** statements are:

```
.PARAMDEX
+   EM(MOPA,TOX)  SIG=2.5
+   EM(MIPC,TOX)  SIG=2.5
+   EM(MIPA,TOX)  SIG=2.5
+   EM(MPN0E,TOX) SIG=2.5
+   EM(MPN4A,TOX) SIG=2.5
+   EM(MPN3A,TOX) SIG=2.5
+   EM(MPN2A,TOX) SIG=2.5
+   EM(MPN1A,TOX) SIG=2.5
+   EM(MONA,TOX)  SIG=2.5
+   EM(MINC,TOX)  SIG=2.5
+   EM(MINA,TOX)  SIG=2.5
+   EM(MNN0D,TOX) SIG=2.5
+   EM(MNN4C,TOX) SIG=2.5
+   EM(MNN4B,TOX) SIG=2.5
```

These statements define a list of independent factors, one for each device. Eldo acts as if private models were created for each device leaving the original model unchanged.

Notes on Statistical Modeling for Discrete Circuits

An important feature of the Integrated Circuit level models is that the designable parameters, such as widths and lengths of transistors, are deterministic in nature and that they are not in the same space as the noise factors. This property distinguishes IC statistical design from the discrete circuit statistical design as explained below.

Consider the following statements extracted from the example *filter.cir* (available in the directory: *$MGC_AMS_HOME/examples/optimizer/*).

```
V1 1 0 DC 0 AC 1
R1 1 2      'R1 + D_R1'
R3 2 0      'R3 + D_R3'
C2 2 INN    'C + D_C'
C3 2 OUT    'C + D_C'
R2 OUT INN 'R2 + D_R2'
Y1 OPAMP1 0 INN OUT 0 PARAM: GAIN='GAIN + D_GAIN' P1='P1 + D_P1'
```

where the circuit parameters are defined as follows:

```
* DETERMINISTIC PARAMETERS
.PARAM R1 = 159K R2 = 3.18MEG R3 = 79.5
+ C = 1U
+ P1 = 10 GAIN = 10E6

* NOISE PARAMETERS
.PARAM D_R1 = AGAUSS(    0,     10.6K,  1)
.PARAM D_R2 = AGAUSS(    0, 0.212MEG,  1)
.PARAM D_R3 = AGAUSS(    0,      5.30,  1)
.PARAM D_C = AGAUSS(    0,    0.067U,  1)
.PARAM D_P1 = AGAUSS(    0,    0.667U,  1)
.PARAM D_GAIN = AGAUSS(  0,   0.667E6,  1)
```

For the resistors the statistical model is expressed by transformations of the form ‘`R + D_R`’ where `R` represents the nominal value and `D_R` is the element tolerance or the random variations associated with the control parameter `R`. These random variables have zero mean. This reflects an important property of discrete RLC circuits that the controllable parameters x and the statistical variables s are in the same space, in other words any circuit performance is a function of random expressions $E(x_i, s_i) = x_i + s_i$.

This property implies that a circuit performance $y(x, s) = y(x + s)$ satisfies the following relation:

$$\frac{\partial}{\partial x}y(x+s) = \frac{\partial}{\partial s}y(x+s)$$

The computation of the sensitivity with respect to the designable variables x will be equivalent to the computation of the sensitivity with respect to the tolerances s, hence in order to screen out the important effects the user can safely investigate only the main effects associated to noise factors.

Notes on Levels Specification

The choice of the levels in the experiment is part of the user's task. In the case of process factors (the noise factors), the **.PARAMDEX** command is directly related to the statistical data defined in the **.PARAM** statement. This information is mainly extracted from the arguments of the command:

```
.PARAM PARAMETER_NAME=EXPRESSION
+ {LOT|DEV|DEVX}[/GAUSS|/UNIFORM] = RVALUE|RVALUE%
```

The value EXPR defines the nominal value x_{nom} or the mean value for the normal and uniform distributions. The value RVALUE represents the *standard deviation* for the normal distribution and the *half-range* for the uniform distribution. The following lines review the basic properties of these important distributions and how the data are used in the **.PARAMDEX** command.

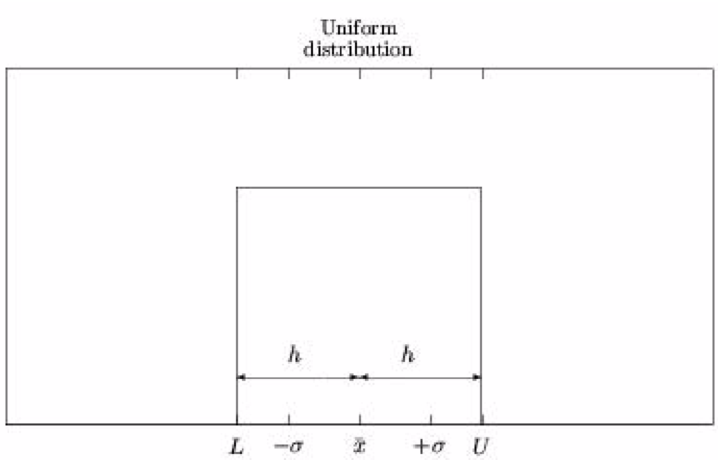
Caution

In Eldo, the definition of the standard deviation (the σ parameter) for the uniform distribution is not correct from the statistical point's of view. A confusion is made between the standard deviation and the range of the distribution. For coherency, the DEX analysis uses the proper definition of this statistical value. This is why the argument **SIG** and **RNG** are available for factors associated to uniform distributions.

Normal and Uniform Distributions

The *uniform distribution* defines equal probability over a given range for a continuous distribution.

Figure 20-2. Common statistics for UNIF($\bar{x}$,h)



The general formula for the probability density function of the uniform distribution is:

$$f(x) = \frac{1}{U - L}$$

for $L \le x \le U$. The common statistics for the uniform distribution are:

$$\bar{x} = \frac{U + L}{2}$$

for the *mean*, and: $\sigma = \sqrt{\frac{(U - L)^2}{12}}$

for the *standard deviation*. The *range* of the distribution is the value $U - L$. It means that for the following definition

```
.PARAM PARAMETER_NAME=EXPRESSION {LOT|DEV|DEVX} /UNIFORM=RVALUE|RVALUE%
.PARAM PARAMETER_NAME={AUNIF|UNIF}(EXPRESSION,RVALUE)
```

the value `RVALUE`, noted h in our formulae, represents half of the range (see Figure 20-2), we will have:

$$\sigma = \frac{h}{\sqrt{3}} \approx 0.6 \times h$$

or when this value h is specified as a percentage,

$$\sigma = \frac{h \times |x_{nom}|}{\sqrt{3}}$$

The general formula for the probability density function of the *normal distribution* is:

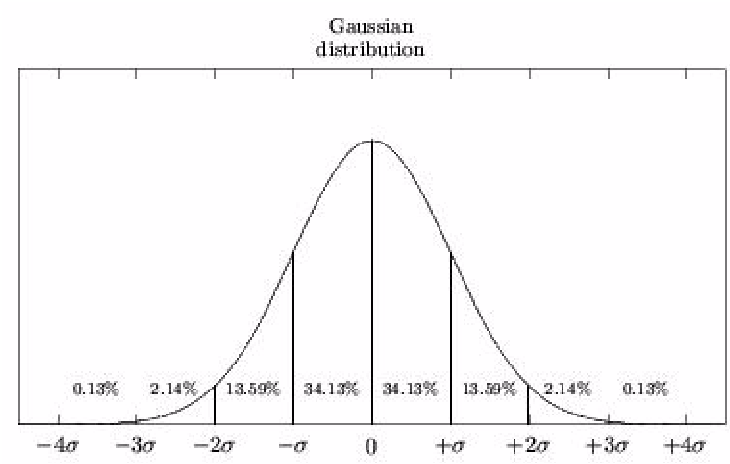
$$f(x) = \frac{e^{-(\bar{x} - \mu)^2 / (2\sigma^2)}}{\sigma\sqrt{2\pi}}$$

where the parameters $\bar{x}$, the mean, and $\sigma > 0$, the standard deviation, are given.

The user has to note that the macro **`GAUSS`**`(NOM_VALUE,REL_VALUE,SIG_COEF)` allows negative values for the standard deviation `SIGMA`, since this value is computed as `NOM_VALUE*REL_VALUE/SIG_COEF`. It means the nominal value can be negative. The **`.DEX`** command will only consider the absolute value of these parameters.

The plot in Figure 20-3 depicts the common statistics of the Gaussian distribution.

Figure 20-3. Percentage of cases in eight portions of the Gaussian distribution

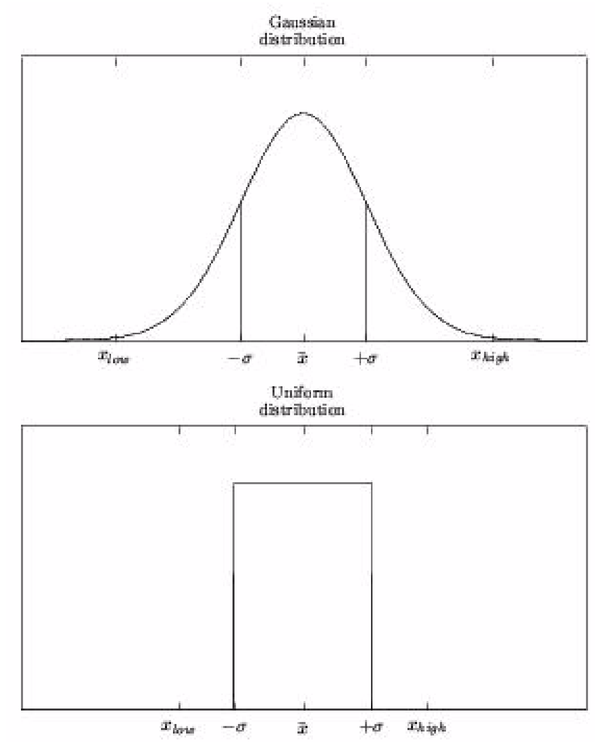


It may help the user in defining the levels of a factor associated to such function. For example, the range $[-\sigma, \sigma]$ represents 34.13% + 34.13% = 68.26% of cases obtained after running the sample process in the Monte Carlo algorithm, and the range $[-3\sigma, 3\sigma]$ covers 2(34.13% + 13.59% + 2.14%) = 99.72% of the cases. It means that the "High" and "Low" levels are the worst case points of the previous ranges.

Levels as Fraction of the Standard Deviation

This represents a straightforward way to specify levels on noise parameters. The high and low levels are defined with $\pm n \cdot \sigma$ where σ is the standard deviation defined on the factor. This situation is depicted in Figure 20-4.

Figure 20-4. Factor levels x_{low} and x_{high} for a circuit parameter x



The syntax is defined as follows:

```
.PARAMDEX FACTOR_NAME [NOISE/] SIG=NSIGMA
```

Note that it is possible to define levels outside of the range of a uniform distribution, the NSIGMA value may be greater than $\sqrt{3}$, we then may have $x_{high} \geq U$ and $x_{low} \leq L$.

This syntax assumes that a uniform or gaussian distribution has been defined on the circuit parameter.

Levels as Fraction of the Half Range of Uniform Distribution

This option gives the possibility to specify levels on noise parameters associated to uniform distribution. The high and low levels are defined with $\pm n \cdot h$ where h is the half range of the uniform distribution on the factor. The command is as follows

```
.PARAMDEX FACTOR_NAME [NOISE/] RNG=NRANGE
```

Note that it is possible to define levels outside of the range of a uniform distribution, the NSIGMA value may be greater than 1.0, we then may have $x_{high} \geq U$ and $x_{low} \leq L$.

Levels are specified with respect to the nominal value

In the cases of noise and control factors, the values of the low level x_{low} and high level x_{high} can be defined explicitly with the following syntaxes:

```
.PARAMDEX FACTOR_NAME [NOISE/] ABS=DELTA
.PARAMDEX FACTOR_NAME [NOISE/] REL=DELTA%


.PARAMDEX FACTOR_NAME [CTRL/] ABS=DELTA [NOM=RVALUE]
.PARAMDEX FACTOR_NAME [CTRL/] REL=DELTA% [NOM=RVALUE]
```

For instance, the low level is $x_{low} = x_{nom} - \Delta$ and $x_{low} = (1 - \Delta) \cdot x_{nom}$ for each type of design factors, and the high level is defined similarly by $x_{low} = x_{nom} + \Delta$ and $x_{high} = (1 + \Delta) \cdot x_{nom}$, where x_{nom} is the nominal value. It is possible to specify a value for the nominal value of a control parameter. This is done with the argument `NOM=RVALUE`.

Examples of Experimental Design

The following table gives a brief description of the basic examples used. These circuits are available in the directory: *$MGC_AMS_HOME/examples/optimizer/*

Example	File name	Description
Passive band pass filter	*bpass.cir*	Basic discrete RLC circuit
Active band pass filter	*filter.cir*	Another discrete circuit
CMOS delay circuit	*cmos_delay.cir*	Basic example with BSIM1 models
CMOS ring oscillator	*cmos_ring.cir*	Another example with BSIM1 models
8-bit CMOS ripple carry adder	*adder.cir*	Yet another example with BSIM1 models

References

1. C. F. Jeff Wu, Michael Hamada. *Experiments: Planning, Analysis, and Parameter Design Optimization*. John Wiley & Sons, 2000.
2. Hedayat, A. and Wallis W.D. *Hadamard Matrices and Their Applications.* Annals of Statistics, Vol. 6, No. 6 (Nov., 1978), pp. 1184-1238.

Chapter 21
Reliability Simulation

Introduction

Reliability models have been implemented in Eldo's UDRM interface (User Defined Reliability Model).

The objective of reliability simulation is to be able to model the gradual damage, which occurs to the devices in a certain design causing degradation in the performance of that design. It is required to evaluate the amount of degradation occurring in a certain period of operation and examine the circuit performance after this period. The modeled damage could follow one or more of several damage mechanisms, which show gradual degradation in device performance. Other mechanisms, which cause sudden and complete damage of the device, (like the oxide breakdown for example) are not targeted in this scope.

The reliability simulations follows the model defined by the user using the UDRM interface. A simple Hot-carrier and NBTI reliability model is provided as an example.

Please refer to *Eldo UDRM User's Manual* for more information.

Running Reliability Simulation in Eldo

Reliability simulation in Eldo can be used with any normal netlist provided that it contains a **`.TRAN`** analysis. This **`.TRAN`** analysis will be the analysis used for all the fresh and aged simulations, and its outputs will be available also for all the fresh and aged simulations. The reliability simulation can be invoked and controlled through the Eldo reliability commands.

The reliability commands are detailed in the *Eldo UDRM User's Manual*, and summarized in Table 21-1.

Table 21-1. Eldo Reliability Commands

Command	Description
.AGE	Age analysis
.AGE_LIB	Define functions for reliability
.AGEMODEL	Reliability model parameter declaration
.SETKEY	Set reliability model key (password)
.USEKEY	Use reliability model key (password)

Chapter 22
Post-Processing Library

Introduction

The Post-Processing Library (PPL) is a tool that allows interfacing between Eldo and a set of predefined, user defined DSP or mathematical functions. User Defined Functions (UDF) are written using the scripting language Tcl. This chapter describes both the commands used from Eldo to call a PPL function and the extensions of the Tcl language for Post-Processing abilities.

A number of example files are provided to illustrate as many configurations as possible, see "Example Files" on page 1260.

General Usage

Registering User Defined Functions inside Eldo

Use Tcl File

```
.USE_TCL FILENAME
```

The **`.USE_TCL`** command (see ".USE_TCL" on page 929) loads the Tcl file, `FILENAME`, into the Eldo Tcl interpreter, making all declared functions to be recognized inside Eldo. Since these functions will have to be identified during the parsing of the netlist, the **`.USE_TCL`** commands must be placed before any call to one of these functions.

After loading the Tcl file, Eldo can use all the functions written in this file as if they were macros. This means that these functions can accept any argument, are defined for the whole netlist, and can return both numbers and/or waves.

Macro-Like Usage of UDF

The following example shows how two resistors get their values directly from a UDF function or through a parameter (the files are *realvalue.cir* and *realvalue.tcl*). See "Example Files" on page 1260 for descriptions and information on where to find these and other example files.

File *realvalue.tcl* listing:

```
# Function which returns a real value
    proc GETVALUE { rname rval } {

# Use native Tcl syntax for expression calculation
```

```
        set newvalue [expr 2*$rval + log($rval)]

        puts "The value of resistor $rname is $newvalue"
        return $newvalue
    }
```

Note

The name of the function *must* be in uppercase, or Eldo will not be able to identify it.

File *realvalue.cir* listing:

```
* This example demonstrates the use of a simple Tcl function
* returning a real value

* use_tcl commands must always be put before the first call
* to a Tcl function. Otherwise, Eldo will not be able to parse
* the expressions using these functions

.use_tcl realvalue.tcl

.param p1={GETVALUE("R1",TEMPER)}
.param p2=1

v1 1 0 sin(0 2 1g)
r1 1 2 p1
r2 2 0 {GETVALUE("R2",p2)}

.tran 1n 10n
.plot tran i(r1)
.end
```

Except for the dump of the value, the Tcl function `GETVALUE` is equivalent to:

```
.defmac GETVALUE(val) = 2*val+log(val)
```

UDF parameters can be real values, strings, waves, macros, UDF or keywords (see “Keyword Parameters” on page 1256). It is important to note that for this kind of use, the UDF may be called for each timestep. Thus wave parameters are not passed to the UDF as objects but as real values.

The example files *samphold.cir* and *samphold.tcl* (see “Example Files” on page 1260) contain another example of UDF. The function uses a Tcl namespace to keep the previous value of a wave, and is called inside a **`.DEFWAVE`** to create a sampled representation of the input wave.

Keyword Parameters

The following keywords can be used in any calls to UDF:

`XAXIS` / **`TIME`** / **`FREQ`**	specifies the current timestamp or frequency
`TEMPER`	specifies the current temperature

The following keywords can only be used in **.CALL_TCL** commands:

ICARLO	specifies the current index of Monte-Carlo run
IRUN	specifies the current step index
IALTER	specifies the current alter index
NBCARLO	specifies the number of Monte-Carlo run
NBRUN	specifies the number of step
NBALTER	specifies the number of **.ALTER** statements

Making Specific Calls to UDF inside Eldo

The **CALL_TCL** command is used to perform a single call to a Tcl function. For details on this command see ".CALL_TCL" on page 559. The example files (see "Example Files" on page 1260) *expressions.cir* and *expressions.tcl* demonstrate how **.CALL_TCL** can be used.

Working with Waveforms inside UDF

Waveforms in Arguments

It is very important to note that in the context of **.CALL_TCL** commands, waves are not real values but vectors, or, in a more general way, objects are generated by Eldo and given to the PPL. The argument that UDF receives is only a reference to an object. Thus it is impossible to use directly these references in mathematical expressions such as:

```
proc WAVE_PLUS { wv_in } {set wv_out [expr $wv_in + $wv_in]
+ return $wv_out}
```

Waveforms in Expressions

To allow operations on waveforms, a set of commands has been registered inside the PPL environment. The following table lists common operators' syntax and the equivalent syntax used inside the PPL.

Table 22-1. C and PPL Syntax

C Syntax	C Example	PPL Syntax	PPL Example
+	a = b + c	add	set a [add $b $c]
-	a = b - c	sub	set a [sub $b $c]
*	a = b * c	mult	set a [mult $b $c]
/	a = b / c	div	set a [div $b $c]
^	a = b ^ c	pow	set a [pow $b $c]

Table 22-1. C and PPL Syntax

C Syntax	C Example	PPL Syntax	PPL Example
fmod	a = fmod(b,c)	mod	set a [mod $b $c]

These commands accept real, complex and/or wave arguments. For example, it is possible to add two waveforms, a waveform and a real, or two reals.

Looking more closely at the Tcl syntax, it is obvious that complex expressions can be very difficult to read.

For example, in common language:

```
a = ((b+c)*(d+e))/(f+g)
```

becomes, in Tcl:

```
set a [div [mult [add $b $c] [add $d $e]] [add $f $g]]
```

Two commands have been defined to simplify the syntax, **`evalExpr`** (see evalExpr) and **`defineVec`** (see defineVec).

The example files, *expressions.cir* and *expressions.tcl*, show the two different syntaxes. See "Example Files" on page 1260 for more information on example files.

evalExpr

Syntax

```
evalExpr expression
```

Description

Evaluates an expression in C-Like syntax. The `evalExpr` command allows evaluation of an expression written using C-like syntax. It can contain both numbers and/or waves with quotes being mandatory. If the expression is purely numerical a single value is returned. Otherwise the result is a vector and is printed using the display command format (display command and output will be described later).

This command is more likely to be used with numerical expressions whereas the `defineVec` command (see "defineVec" on page 1260) is for wave calculations.

Example

```
proc EVAL_EXPR { wv_in } {
    set a 3
    set numerical [evalExpr "2*$a + 4"]
    puts "numerical = $numerical"
    set vector [evalExpr "2*$wv_in + 4"]
    puts "vector = $vector"
}
```

A call to this function from Eldo command:

```
.call_tcl tran where=END eval_expr(v(1))
```

produces the following output:

```
numerical = 10.00000000000000
vector = { 0 0.00000000000000 4.00000000000000 +
0.00000000000000j }
 { 1 0.00000000002000 4.50133293425722 + 0.00000000000000j }
 { 2 0.00000000003900 4.97030313987119 + 0.00000000000000j }
 { 3 0.00000000007699 5.86034757905359 + 0.00000000000000j }
 { 4 0.00000000015297 7.27942102760290 + 0.00000000000000j }
 { 5 0.00000000022966 7.96738275669581 + 0.00000000000000j }
 { 6 0.00000000025000 8.00000000000000 + 0.00000000000000j }
 { 7 0.00000000029068 7.87006296906364 + 0.00000000000000j }
 { 8 0.00000000035325 7.18743864881722 + 0.00000000000000j }
```

defineVec

Syntax

```
defineVec <output_object> expression
```

Description

Defines a PPL object using C-Like syntax. Like the `evalExpr` command (see “evalExpr” on page 1259), `defineVec` allows evaluation of an expression written using C-like syntax but the result is stored in a PPL object named `<output_object>` rather than displayed. An error occurs if an object using the same name already exists in the PPL.

This command is more likely to be used with wave calculations, whereas the `evalExpr` command is for numerical expressions.

Example

```
proc DEFINE_EXPR { wv_in } {
    set a 3
    defineVec numerical "2*$a+4"
    puts "numerical = [display numerical]"

    defineVec vector "$wv_in+4"
    puts "vector = [display vector]"
    return vector
}
```

A call to this function from the Eldo command:

```
.call_tcl tran where=END define_expr(v(1))
```

produces the same output as `evalExpr` but the object `vector` can be reused in any expression or returned to Eldo in order to be dumped in the output file.

Caution

It is very important to note that the return value of this function is not a Tcl variable but directly `vector`.

Example Files

The following example files are provided in your installed directory:

$MGC_AMS_HOME/examples/ppl

- *realvalue.cir* and *realvalue.tcl*

  demonstrates how to use UDF in a macro-like way to define a resistor value.

- *samphold.cir* and *samphold.tcl*

  demonstrates how to use UDF in a macro-like way to define defwaves, and namespace in Tcl.

- *expressions.cir* and *expressions.tcl*

  demonstrates how to use **`.CALL_TCL`** commands and how to work with expressions inside the UDF.

- *tcl_wave.cir* and *expressions.tcl*

  demonstrates how to use the **`.TCL_WAVE`** command.

- *wave_info.cir* and *wave_info.tcl*

  demonstrates how to use wave informations commands and loops in Tcl.

- *wave_meas.cir* and *wave_meas.tcl*

  demonstrates how to use measurement functions in UDF.

- *asciigen.cir* and *asciigen.tcl*

  demonstrates how to use several **`.CALL_TCL`** commands to dump waves in an ASCII output file.

Built-In Waveform Functions

Table 22-2. Built-In Waveform Functions

Eldo Functions				
ABS	ACOS	ACOSH	ACOT	ACOTH
ASIN	ASINH	ATAN	ATANH	AVG
COS	COSH	COT	COTH	DB
DEG	DELAY	DERIVE	DRV	EXP
FALLTIME	GMARGIN	HISTOGRAM	IMAG	INTEG
INTEGRAL	INTERSECT	INVDB	LOG	LOG10
MAG	MAX	MIN	PHASE	PHMARGIN
RAD	REAL	RELATION	RISETIME	RMS
SAMPLE	SETTLINGTIME	SIN	SINH	SQR
SQRT	SUM	TAN	TANH	VMAX
VMIN	WINDOW	XVAL	XWAVE	YVAL

Table 22-2. Built-In Waveform Functions

Eldo RF Functions				
COMPRESS	IIPXIIPX	OIPX	TPD	TPDDD
TPDDU	TPDUDTPDUD	TPDUU	XCOMPRESS	

In the following descriptions, the syntax is shown immediately after the function name, then a short description, then, if applicable, descriptions of the arguments. `wv_in` is the input waveform the function is acting upon. `wv_out` is the resulting output waveform.

Eldo Functions

ABS

```
set wv_out [abs $wv_in]
```

Absolute wave.

ACOS

```
set wv_out [acos $wv_in]
```

Inverse trigonometric wave function.

ACOSH

```
set wv_out [acosh $wv_in]
```

Inverse hyperbolic wave function.

ACOT

```
set wv_out [acot $wv_in]
```

Inverse trigonometric wave function.

ACOTH

```
set wv_out [acoth $wv_in]
```

Inverse hyperbolic wave function.

ASIN

```
set wv_out [asin $wv_in]
```

Inverse trigonometric wave function.

ASINH

```
set wv_out [asinh $wv_in]
```

Inverse hyperbolic wave function.

ATAN

```
set wv_out [atan $wv_in]
```

Inverse trigonometric wave function.

ATANH

```
set wv_out [atanh $wv_in]
```

Inverse hyperbolic wave function.

AVG

```
set out_value [avg $wv_in]
set out_value [avg $wv_in $lower_bound $upper_bound]
```

Average value of a wave.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

COS

```
set wv_out [cos $wv_in]
```

Trigonometric wave function.

COSH

```
set wv_out [cosh $wv_in]
```

Hyperbolic wave function.

COT

```
set wv_out [cot $wv_in]
```

Trigonometric wave function.

COTH

```
set wv_out [coth $wv_in]
```

Hyperbolic wave function.

DB

```
set wv_out [db $wv_in]
```

Conversion dB magnitude. See also INVDB.

DEG

```
set wv_out [deg $wv_in]
```

Conversion degrees. See also RAD.

DELAY

```
set wv_out [delay $wv_in $delay_value]
```

Creates a delayed wave.

DERIVE

```
set out_value [derive $wv_in $xvalue]
```

Derivative of a wave at an x value.

DRV

```
set wv_out [drv $wv_in]
```

Derivative of a wave with respect to the x (or scale) variable.

EXP

```
set wv_out [exp $wv_in]
```

Wave exponential.

FALLTIME

```
set out_value [falltime $wv_in $lower_thr $upper_thr]
set out_value [falltime $wv_in $lower_thr $upper_thr $after]
```

Fall time of a wave, the result is of unit {x_unit}.

- `lower_thr`

  Lower threshold.
- `upper_thr`

  Upper threshold.
- `after`

  First transition between lower and upper thresholds is treated after this x value.

GMARGIN

```
set out_value [gmargin $wv_in1 $wv_in2]
```

Gain margin is the linear gain distance to 1.0. See gain margin extraction example.

HISTOGRAM

```
set wv_out [histogram $wv_in $intervals]
```

```
set wv_out [histogram $wv_in $intervals $sample]
set wv_out [histogram $wv_in $intervals $lower_bound $upper_bound]
set wv_out [histogram $wv_in $intervals $lower_bound $upper_bound $sample]
```

Generates a histogram of the input wave showing the waves magnitude probability density distribution. The input wave is first sampled to equidistant points if the optional argument <$sample> is set.

- `intervals`

  the number of histogram intervals

- `sample`

  if 0 or not specified: don't sample

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

IMAG

```
set wv_out [imag $wv_in]
```

Imaginary part of a magnitude and phasewave pair (magnitude: linear, phase: in degrees). See also REAL.

INTEG

```
set out_value [integ $wv_in]
set out_value [integ $wv_in $lower_bound $upper_bound]
```

Definite integral of the input wave with respect to the x (or scale) variable, global integral or interval integral the result has the unit {wave_unit}*{x_unit}.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

INTEGRAL

```
set wv_out [integral $wv_in $first_y_value]
```

Indefinite integral of the input wave with respect to the x-axis variable. The result is a waveform.

- `first_y_value`

  Integration constant

INTERSECT

```
set out_vector [intersect $wv_in1 $wv_in2]
set out_vector [intersect $wv_in1 $wv_in2 $lower_bound $upper_bound]
```

Get the intersection point (s) of two waves.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

INVDB

```
set wv_out [invdb $wv_in]
```

Conversion linear magnitude. See also DB.

LOG

```
set wv_out [log $wv_in]
```

Wave logarithm.

LOG10

```
set wv_out [log10 $wv_in]
```

Wave logarithm.

MAG

```
set wv_out [mag $wv_in]
```

Linear magnitude from real and imaginary part.

MAX

```
set wv_out [max $wv_in1 $wv_in2]
```

Maximum wave of two waves (max is evaluated point-by-point).

MIN

```
set wv_out [min $wv_in1 $wv_in2]
```

Minimum wave of two waves (min is evaluated point-by-point).

PHASE

```
set wv_out [phase $wv_in]
```

Phase (in degrees) from real and imaginary part.

PHMARGIN

```
set out_value [phmargin $wv_in1 $wv_in2]
```

Phase margin is the phase distance to -180 degrees. See phase margin extraction example.

RAD

```
set wv_out [rad $wv_in]
```

Conversion radians. See also DEG.

REAL

```
set wv_out [real $wv_in]
```

Real part of a magnitude and phasewave pair (magnitude: linear, phase: in degrees). See also IMAG.

RELATION

```
set wv_out [relation $wv_in1 $wv_in2 $operator]
```

Generates a wave from the point-by-point comparison of two waves. It returns 1 whenever operator is 1 and wv_in1 > wv_in2, or operator is 0 and wv_in1 == wv_in2, or operator is -1 and wv_in1 < wv_in2. Otherwise it returns 0.

RISETIME

```
set out_value [risetime $wv_in $lower_thr $upper_thr]
set out_value [risetime $wv_in $lower_thr $upper_thr $after]
```

Rise time of a wave, the result is of unit {x_unit}.

- `lower_thr`

  Lower threshold.

- `upper_thr`

  Upper threshold.

- `after`

  First transition between lower and upper thresholds is treated after this x value.

RMS

```
rms=sqrt ((1/ (x_max-x_min))*integ(wave^2)).
set out_value [rms $wv_in]
set out_value [rms $wv_in $lower_bound $upper_bound]
```

Root mean square value of a wave.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

SAMPLE

```
set wv_out [sample $wv_in $sample_time]
set wv_out [sample $wv_in $sample_time $lower_bound $upper_bound $sample]
```

Creates a sampled wave.

- `sample_time`

  sampling period
- `lower_bound`

  x value after which sampling starts
- `upper_bound`

  x value after which sampling stops

SETTLINGTIME

```
set out_value [settlingtime $wv_in $yvalue $eps]
set out_value [settlingtime $wv_in $yvalue $eps $lower_bound $upper_bound]
```

Time required for the input wave to settle within a certain limit around the final value.

- `yvalue`

  Value to reach.
- `eps`

  Tolerance value (yvalue +/- eps/2).
- `lower_bound`

  x value after which measurement starts
- `upper_bound`

  x value after which measurement stops

SIN

```
set wv_out [sin $wv_in]
```

Trigonometric wave function.

SINH

```
set wv_out [sinh $wv_in]
```

Hyperbolic wave function.

SQR

```
set wv_out [sqr $wv_in]
```

Wave square.

SQRT

```
set wv_out [sqrt $wv_in]
```

Wave square root.

SUM

```
set out_value [sum $wv_in]
set out_value [sum $wv_in $lower_bound $upper_bound]
```

Sums up the real part of a vector.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

TAN

```
set wv_out [tan $wv_in]
```

Trigonometric wave function.

TANH

```
set wv_out [tanh $wv_in]
```

Hyperbolic wave function.

VMAX

```
set out_value [vmax $wv_in]
set out_value [vamx $wv_in $lower_bound $upper_bound]
```

Maximum value of a wave.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

VMIN

```
set out_value [vmin $wv_in]
set out_value [vmin $wv_in $lower_bound $upper_bound]
```

Minimum value of a wave.

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

WINDOW

```
set wv_out [window $wv_in $lower_bound $upper_bound]
```

Selects a window (a, b) out of a wave, the points at the interval bounds are interpolated.

- `lower_bound`, `upper_bound`

  Begin and end of the wave window to select.

XVAL

```
set out_vector [xval $wv_in $yvalue]
set out_vector [xval $wv_in $yvalue $slope]
set out_vector [xval $wv_in $yvalue $slope $lower_bound $upper_bound]
```

All x (or scale) values for one wave (or y) value.

- `slope`

  Find only x values, where wave crosses y with slope:

  slope > 0 Positive slope

  slope < 0 Negative slope

  slope = 0 Positive and negative slope

- `lower_bound`

  x value after which measurement starts

- `upper_bound`

  x value after which measurement stops

XWAVE

```
set wv_out [xwave $wv_in]
```

Creates a wave with y = x values.

YVAL

```
set out_value [yval $wv_in $xvalue]
```

Wave value at x (or scale) value (linear interpolation to get the exact value).

RF Functions

COMPRESS

```
set out_value [compress $wv_in $val]
```

Extracts the Y-axis value of the wave at the point where the difference between the actual value of the wave and the linear extrapolation of the wave based on the computed slope value becomes greater than val.

IIPX

```
set out_value [iipx $wv_in $wave_out $freq1 $freq2]
```

Returns the input referred intercept point of order x from the value of the circuit input and output, wv_in and wave_out respectively. wv_in and wave_out must be in dB or dBm. The intercept order is directly calculated from the inter modulation of freq1 and freq2.

OIPX

```
set out_value [oipx $wv_in $freq1 $freq2]
```

Returns the output referred intercept point of order x from the value of the circuit output wave, this must be in dB or dBm. The intercept order is directly calculated from the inter modulation of freq1 and freq2.

TPD

```
set out_value [tpd $wv_in1 $wv_in2]
set out_value [tpd $wv_in1 $wv_in2 $vth]
set out_value [tpd $wv_in1 $wv_in2 $vthin $vthout]
set out_value [tpd $wv_in1 $wv_in2 $vthin $vthout $before $after]
set out_value [tpd $wv_in1 $wv_in2 $vthin $vthout $before $after $occur]
```

Returns the propagation delay between wv_in1 and wv_in2.

Note

For a complete description of TPD and its parameters, see ".EXTRACT" on page 637.

- `vth`

  Voltage defining a threshold or starting point.

- `vthin`

  Voltage defining a threshold or starting point.

- `vthout`

  Voltage defining a threshold or starting point.

- `before`

  Causes the function to be performed only if $TIME < val$.

- `after`

  Causes the function to be performed only if $TIME > val$.

- `occur`

  Computes the function for the VALth occurrence of the event.

TPDDD

Returns the propagation delay(TPD) with WAVE1 and WAVE2 both falling.

TPDDU

Returns the propagation delay(TPD) with WAVE1 falling, WAVE2 rising.

TPDUD

Returns the propagation delay(TPD) with WAVE1 rising, WAVE2 falling.

TPDUU

Returns the propagation delay(TPD) with WAVE1 and WAVE2 both rising.

XCOMPRESS

```
set out_value [xcompress $wv_in $val]
```

Extracts the X-axis value of the wave at the point where the difference between the actual value of the wave and the linear extrapolation of the wave based on the computed slope value becomes greater than val.

Built-In DSP Functions

Table 22-3. Built-In DSP Functions

AUTOCOR	CONV	CORRELO	CT	FFT
HARMONICS	HDIST	IFFT	PERIODO	PSD
SAMPLER	SNR			

In the following descriptions, the syntax is shown immediately after the function name, then a short description, then, if applicable, descriptions of the arguments.

AUTOCOR

```
set wv_out [autocor $wv_in $method]
set wv_out [autocor $wv_in $method $nbpts]
```

Calculates the Auto Correlation Function (AF) of a waveform.

The AF of a signal waveform is an average measure of its time domain properties and therefore especially relevant when the signal is random.

There are two methods available for calculating the auto correlation, namely the Correlogram and Periodogram methods.

For more explanation refer to the *EZwave User's Manual*.

- `method`

  Select the AF calculation method.

  0 for correlogram (default)

  1 for periodogram

- `nbpts`

  Number of points for the AF result.

CONV

```
set wv_out [conv $wv_in1 $wv_in2 $nptsa $nptsb $fs]
```

Calculates the convolution of two signals. For more explanation refer to the *EZwave User's Manual*.

- `nptsa`

  Number of points for wave1.

- `nptsb`

  Number of points for wave2.

- `fs`

  Sampling Frequency.

CORRELO

```
set wv_out [correlo $wv_in $tstart $tstop $fs $nbpts $padding
+ $sample $fmin $fmax $normalized $ncor $nauto $npsd]
```

Calculates the Power Spectral Density using correlogram method. For more explanation refer to the *EZwave User's Manual*.

- `tstart`

  Start time for the signal.

- `tstop`

  Stop time for the signal.

- `fs`

  Sampling frequency.

- `nbpts`

  Number of sampling points.

- `padding`

  Selects the padding zeros type:

  0 No Padding

  1 Padding at the end

  2 Padding at the start

  3 Center wave

- `sample`

  Selects the sampling type:

  0 No sampling

  1 Interpolation

  2 Sample and Hold

- `fmin`

  Starting frequency used inside the FFT result window.

- `fmax`

  Last frequency used inside the FFT result window.

- `normalized`

  0 No normalization

  1 All real and imaginary parts of the result are divided by Nbpts/2 except for the first point, which is divided by Nbpts.

- `ncor`

  Number of auto correlation points used for the PSD computation. Default value is Nauto.

- `nauto`

  Number of points for the auto correlation result. Default value is Nbpts. Necessary condition is Nauto 2 >= 2

- `npsd`

  Number of points for the PSD result. Default value is Ncorr/2 + 1. Necessary condition is npsd >= Ncorr/2 + 1.

CT

```
set wv_out [ct $wv_in $tstart $tstop $fs $nbpts $padding $sample $fmin
$fmax $normalized $nzoom]
```

Calculates the Chirp Transformed (CT) wave. For more explanation refer to the *EZwave User's Manual*.

- `tstart`

  Start time for the signal.
- `tstop`

  Stop time for the signal.
- `fs`

  Sampling frequency.
- `nbpts`

  Number of sampling points.
- `padding`

  Selects the padding zeros type:

  0 No Padding

  1 Padding at the end

  2 Padding at the start

  3 Center wave
- `sample`

  Selects the sampling type:

  0 No sampling

  1 Interpolation

  2 Sample and Hold
- `fmin`

  Starting frequency used inside the FFT result window.
- `fmax`

  Last frequency used inside the FFT result window.
- `normalized`

  0 No normalization

  1 All real and imaginary parts of the result are divided by Nbpts/2 except for the first point, which is divided by Nbpts.
- `nzoom`

  Number of points for the zooming.

FFT

```
set wv_out [fft $tstart $tstop $fs $nbpts $padding $sample $fmin $fmax
$normalized]
```

Calculates the Fast Fourier Transform (FFT) of a wave. FFT is the fastest and most efficient available algorithm for computing the DFT.

- `tstart`

  Start time for the signal.

- `tstop`

  Stop time for the signal.

- `fs`

  Sampling frequency.

- `nbpts`

  Number of sampling points.

- `padding`

  Selects the padding zeros type:

  0 No Padding

  1 Padding at the end

  2 Padding at the start

  3 Center wave

- `sample`

  Selects the sampling type:

  0 No sampling

  1 Interpolation

  2 Sample and Hold

- `fmin`

  Starting frequency used inside the FFT result window.

- `fmax`

  Last frequency used inside the FFT result window.

- `normalized`

  0 No normalization

  1 All real and imaginary parts of the result are divided by Nbpts/2 except for the first point, which is divided by Nbpts.

HARMONICS

```
set wv_out [harmonics $wv_in $fundf]
set wv_out [harmonics $wv_in $fundf $fmin $fmax]
```

Calculates harmonics inside the interval [fmin, fmax] then generates the harmonics wave.

- `fundf`

  Fundamental Frequency.

- `fmin`, `fmax`

  frequency band that should be taken for the computation.

HDIST

```
set out_value [hdist $wv_in $fundf]
set out_value [hdist $wv_in $fundf $fmin $fmax]
```

Calculates the Total Harmonic Distortion (THD) of a signal.

- `fundf`

  Fundamental frequency.

- `fmin`, `fmax`

  Frequency band that should be taken for the computation.

IFFT

```
set wv_out [ifft $wv_in]
```

Calculates the Inverse Fast Fourier Transform.

PERIODO

```
set wv_out [periodo $wv_in $tstart $tstop $fs $nbpts $padding
+ $sample $fmin $fmax $normalized $ncor $nauto $npsd]
```

Calculates the Power Spectral Density using periodogram method. For more explanation refer to the *EZwave User's Manual*.

- `tstart`

  Start time for the signal.

- `tstop`

  Stop time for the signal.

- `fs`

  Sampling frequency.

- `nbpts`

  Number of sampling points.

- `padding`

  Selects the padding zeros type:

  0 No Padding

  1 Padding at the end

  2 Padding at the start

  3 Center wave

- `sample`

  Selects the sampling type:

  0 No sampling

  1 Interpolation

  2 Sample and Hold

- `fmin`

  Starting frequency used inside the FFT result window.

- `fmax`

  Last frequency used inside the FFT result window.

- `normalized`

  0 No normalization

  1 All real and imaginary parts of the result are divided by Nbpts/2 except for the first point, which is divided by Nbpts.

- `ncor`

  Number of auto correlation points used for the PSD computation. Default value is Nauto.

- `nauto`

  Number of points for the auto correlation result. Default value is Nbpts. Necessary condition is Nauto 2 >= 2

- `npsd`

  Number of points for the PSD result. Default value is Ncorr/2 + 1. Necessary condition is npsd >= Ncorr/2 + 1.

PSD

```
set wv_out [psd $wv_in $method]
set wv_out [psd $wv_in $method $nbpts]
```

Calculates the Power Spectral Density (PSD) of a waveform.

The PSD of a signal waveform is an average measure of its time domain properties and therefore especially relevant when the signal is random.

There are two methods available for calculating the PSD, namely the Correlogram and Periodogram methods.

For more explanation refer to the *EZwave User's Manual*.

- `method`

  Select the PSD calculation method.

  0 for correlogram (default)

1 for periodogram

- `nbpts`

  Number of points for the PSD result.

SAMPLER

```
set wv_out [sampler $tstart $tstop $fs $nbpts $padding $wtype $wparam]
```

Generates a sampled wave using some types of windowing.

- `tstart`

  Start time for the signal.

- `tstop`

  Last time for the signal.

- `fs`

  Sampling frequency.

- `Nbpts`

  Number of sampling points.

- `padding`

  Selects the padding zeros type:

  0 No Padding

  1 Padding at the end

  2 Padding at the start

  3 Center wave

- `wtype`

  Window type:

  0 Rectangular (no wparam)

  1 Hamming (no wparam)

  2 Hanning (Alpha= w_param, default is 0.5)

  3 Parzen (no wparam)

  4 Welch (no wparam)

  5 Blackman (no wparam)

  6 Blackman7 (no wparam)

  7 Bartlett (no wparam)

  8 Kaiser (Beta= wparam, default is 10.056)

  9 Klein (no wparam)

10 Tukey (no wparam)

11 Dolph Chebyshev (Alpha=w_param, default: 3.0)

- `wparam`

  Window optional parameter.

SNR

```
set out_value [snr $wv_in $fmin $fmax $freq_list]
```

Calculates the Signal to Noise Ratio in dBs of a noisy wave by using a specific frequency list.

- `fmin`, `fmax`

  Frequency band that should be taken for the computation.

- `freq_list`

  List of frequencies which should be used in the computation.

Accessing Waves Inside an External Database

In the following descriptions, the syntax is shown immediately after the function name, then a short description, then, if applicable, descriptions of the arguments.

LOAD_FILE

```
load_file $filename
```

Loads a *cou* file in memory. This command returns a list of run identifiers. These identifiers are labelled RunX where X is an absolute counter starting from 0.

Examples

If a file *test.cou* contains one run:

```
puts "[load_file test.cou]"
```

Implies “Run0” will be used as the run identifier label.

If a file *test2.cou* contains three runs:

```
puts "[load_file test2.cou]"
```

Implies “Run0 Run1 Run2” will be used as the run identifier labels.

If both commands are called in the same UDF:

```
puts "[load_file test.cou]"
```

Implies “Run0” will be used as the run identifier label.

```
puts "[load_file test2.cou]"
```

Implies “Run1 Run2 Run3” will be used as the run identifier labels.

These identifiers are used to identify waves. For example if file *test.cou* contains waves v(1) and i(r1), their internal names inside the PPL will be Run0::v(1) and Run0::i(r1). These names can be used in all expressions.

DISP_FILE

```
disp_file
disp_file $identifier
```

Displays the list of runs inside the PPL or the list of waves inside a given run.

Example

Considering that *test.cou* contains one run and *test2.cou* contains three runs,

```
puts "[load_file test.cou]"
```

Implies “Run0” will be used as the run identifier label.

```
puts "[load_file test2.cou]"
```

Implies “Run1 Run2 Run3” will be used as the identifier labels.

```
puts "[disp_file]"
```

Implies “Run0 Run1 Run2 Run3” will be used as the identifier labels.

```
puts "[disp_file Run0]"
```

Implies “V(1) I(R1)” will be used as the identifier labels.

UNLOAD_FILE

```
unload_file all
unload_file $identifier
```

Removes all or a single run from memory.

- `identifier`

  a valid run identifier or keyword “all” to remove all runs.

Miscellaneous Commands

In the following descriptions, the syntax is shown immediately after the function name, then a short description, then, if applicable, descriptions of the arguments.

DISPLAY

```
display $ppl_object
```

Writes the content of PPL objects to the standard output.

- `ppl_object`

A reference to a PPL object. Usually, references for waves start with WV and references for numbers start with DB.

GETSIZE

getsize $wv_in

Returns the number of points in a wave.

GETPOINT

getpoint $wv_in $ptindex

Returns the x value, real and imaginary parts of a wave point.

- `ptindex`

  Index of a point (between [0; size[).

SETPOINT

setpoint $wv_in $ptindex $xvalue $yvalue $yimgvalue

Modifies a wave point value.

- `ptindex`

  Index of a point (between [0; size]).

- `xvalue`

  New x-axis value.

- `yvalue`

  New real part value.

- `yimgvalue`

  New imaginary part value.

CREATEVECTOR

createVector vectorName $xlist $ylist

Creates a vector from a list of (x,y) values.

You must first load a *.cou* file prior to using the createVector command. It can be any *.cou* file.

- `vectorName`

  the name of the PPL object which will represent the vector in the library.

- `xlist`

  A list of ordered x values.

* `ylist`

  A list of values (same length as xlist).

Example

```
load_file dummy.cou
createVector essai "0 1e-9 2e-9 3e-9 4e-9" "1.1 2.2 3.3 4.4 5.5"
```

Chapter 23
IBIS Models Support in Eldo

Introduction

I/O Buffer Information Specification (IBIS) is an ANSI/EIA-656 standard that is developing a specified industry standard method to electronically transport input/output buffers modeling data between semiconductor vendors, EDA tool vendors and end customers.

Eldo supports IBIS version IBISv4.2.

This chapter provides a brief introduction and background to IBIS, lists the current state of IBIS support in Eldo, and provides the syntax and its description for simulating either individual IBIS buffers or entire components. General notes and options related to the IBIS support in Eldo are also described, and some illustrative examples provided. Finally, the chapter ends with some tutorials. It is not intended to provide a detailed description of the IBIS standard in this chapter.

IBIS Background

IBIS was originally developed by Intel® Corporation in the early 1990s. With the participation of more companies and industry members, IBIS Open Forum was created to promote the IBIS development and to make sure that standard exists. In 1995 the IBIS Open Forum teamed with the EIA. Since then the EIA/IBIS Open Forum oversaw all technical developments of IBIS.

The IBIS committee provides a new version every one or two years. Each version adds more features that the EDA tool vendors should support, and describes how the related input data is formatted. The latest IBIS standard is IBIS version 4.2 which was ratified by the EIA IBIS Open Forum on June 2, 2006 and formally ratified as ANSI/EIA-656-B on September 1, 2006.

IBIS is a behavioral model that describes the electrical characteristics of the I/O buffers using V/I and V/T tables. It has the advantage of not showing any proprietary information. Moreover, the use of behavioral simulation, instead of transistor level simulation, helps in accelerating the signal integrity analysis and the overall design cycle. It is important to note that IBIS does not provide the models themselves but describes how the input data to the EDA tool is formatted. An IBIS file contains data not only about individual buffers but also about an entire component (IC) including pin-to-buffer mapping and package parasitics.

Using IBIS Models in Eldo

Using IBIS in Eldo is similar to using other SPICE elements, such as transistors. You specify a name for the buffer, the nodes to which the buffer is connected to the rest of the circuit, and parameters to refer to a model for the buffer and the IBIS file where the model is located. In Eldo you specify the name of the IBIS element with the prefix **_IO_**.

Eldo can simulate IBIS devices either individually or in the overall component. Eldo supports external models/circuits written in SPICE or Verilog-A. External models/circuits written in VHDL-AMS and Verilog-AMS can be simulated using Questa ADMS. Eldo is compatible with HSPICE syntax, under option compat, except for multilingual model Support.

See "IBIS Support in Eldo" on page 1301 for further information.

IBIS Resources on the Web

The best way to learn about IBIS is to review the IBIS specification, published papers and sample models. Extensive IBIS information is available on the web as follows:

- Official IBIS Open Forum website

  http://www.eigroup.org/ibis/

- Mentor Graphics IBIS Modeling Resources

  http://www.mentor.com/products/pcb/expedition/modeling_resources.cfm

Through these web sites you can: receive technical support; review articles, presentations and FAQs; find links to most public IBIS models from semiconductor and connector supplier sources; get free tools and documents that help create an IBIS model; and retrieve the IBIS specifications.

IBIS File Types and Structures

There are three supported file types within the IBIS modeling framework:

- IBIS files, **.ibs*

  The IBIS file is the main file that contains the basic information about the component(s) and their internal buffers. The *.ibs* file also contains information about the package electrical characteristics (package model).

- Package files, **.pkg*

  Advanced package models can be either placed inside the *.ibs* file itself or in a package file (*.pkg*).

- Electrical board description files, **.ebd*

  An electrical board description file (*.ebd* file) is defined to describe the connections of a board level component between the board pins and its components on the board.

The IBIS files contain the same basic information and can be thought of as having three main sections:

- Header section—contains information about the IBIS version, the file name, and the process or organization responsible for the information
- Component section—describes the component name, pinout, pin to buffer mapping, and package electrical characteristics
- Model section—describes the behavior of each unique buffer used in a component

An IBIS *.ibs* file consists of:

- File Header Section
- [Component]
- [Model]
- [Define Package Model]
- [End]

A package *.pkg* file consists of:

- File Header Section
- [Define Package Model]
- [End]

An electrical board description file *.ebd* file consists of:

- File Header Section
- [Begin Board Description]
- [End]

Checking IBIS Files with the Golden Parser

The golden parser is a syntax-checking program that aids the development and verification of IBIS data files. It is available in executable format, compiled for a variety of operating systems, free of charge through the IBIS website (http://www.eigroup.org/ibis/tools.htm).

The latest version of this parser, ibischk4 v4.2.2, is integrated inside Eldo. The golden parser produces warnings and errors which are displayed in the Eldo output by default when an IBIS file is incorporated in the simulation netlist.

IBIS Device Standards

Buffers

IBIS is a modeling technique that provides a simple table-based buffer model for semiconductor devices. IBIS models can be used to characterize I/V output curves, rising/falling transition waveforms, and package parasitic information of the device. There are four types of buffer in IBIS:

- Single-ended buffer
- Pseudo-differential buffer
- True differential buffer
- Series buffer

IBIS also defines a global list of port names with fixed functionality that can be applied to all types of buffers. Eldo uses the same port naming and definition for the buffer equivalent subcircuit. Table 23-1 lists the port names and their descriptions. Port names starting with “A” are used to define analog ports while those starting with “D” are used to define digital ports.

Table 23-2 lists all the available buffer types, assigns a number for each type, lists nodes used by buffer type, and provides the min/max number needed for each buffer type. “[ ]” is used to denote optional nodes. Pseudo-differential buffers are recognized as a separate buffer type by the IBIS standard and are constructed from two single-ended buffers. Therefore, pseudo-differential buffer nodes can be obtained from their single-ended version by replacing “a_signal” with “a_signal_pos” and “a_signal_neg”. PN is used to denote the power nodes, which are:

- **a_pcref - a_gcref** for Input, Input_ECL, Terminator, and Input_diff buffers.
- **a_pcref - a_gcref - a_puref - a_pdref** for other types.

Table 23-1. List of Predefined Port Names and their Descriptions

Port Name	Description
A_signal	I/O signal port for a model unit
A_signal_pos	Non-inverting port of a differential model
A_signal_neg	Inverting port of a differential model
A_pos	Non-inverting port for series or series switch models
A_neg	Inverting port for series or series switch models
A_puref	Voltage reference port for pull-up structure

Table 23-1. List of Predefined Port Names and their Descriptions

Port Name	Description
A_pcref	Voltage reference port for power clamp structure
A_pdref	Voltage reference port for pull-down structure
A_gcref	Voltage reference port for ground clamp structure
A_gnd	Global reference voltage port
D_drive	Digital input to a model unit
D_enable	Digital enable for a model unit
D_receive	Digital receive port of a model unit, based on data on A_signal (and/or A_signal_pos and A_signal_neg)
D_switch	Digital input for control of a series switch model

Table 23-2. Buffer Types and Port Names Used

Type	Number	Nodes (in order)	Min/Max
Input	1	a_signal - [d_receive] - [PN]	1/4
Output	2	a_signal - d_drive - [PN]	2/6
I/O	3	a_signal - d_drive - d_enable - [d_receive] - [PN]	3/8
3-state	4	a_signal - d_drive - d_enable - [PN]	3/7
Open_drain	5	a_signal - d_drive - [PN]	2/6
I/O_open_drain	8	a_signal - d_drive - d_enable - [d_receive] - [PN]	3/8
Open_sink	7	a_signal - d_drive - [PN]	2/6
I/O_open_sink	8	a_signal - d_drive - d_enable - [d_receive] - [PN]	3/8
Open_source	9	a_signal - d_drive - [PN]	2/6
I/O_open_source	10	a_signal - d_drive - d_enable - [d_receive] - [PN]	3/8
Input_ECL	11	a_signal - [d_receive] - [PN]	1/4
Output_ECL	12	a_signal - d_drive - [PN]	2/6
I/O_ECL	13	a_signal - d_drive - d_enable - [d_receive] - [PN]	3/8
3-state_ECL	14	a_signal - d_drive - d_enable - [PN]	3/7
Series	15	a_pos - a_neg	2
Series_switch	16	a_pos - a_neg - [d_switch]	2/3
Terminator	17	a_signal - [PN]	1/3
Input_diff	18	a_signal_pos - a_signal_neg - [d_receive] - [PN]	2/5

Table 23-2. Buffer Types and Port Names Used

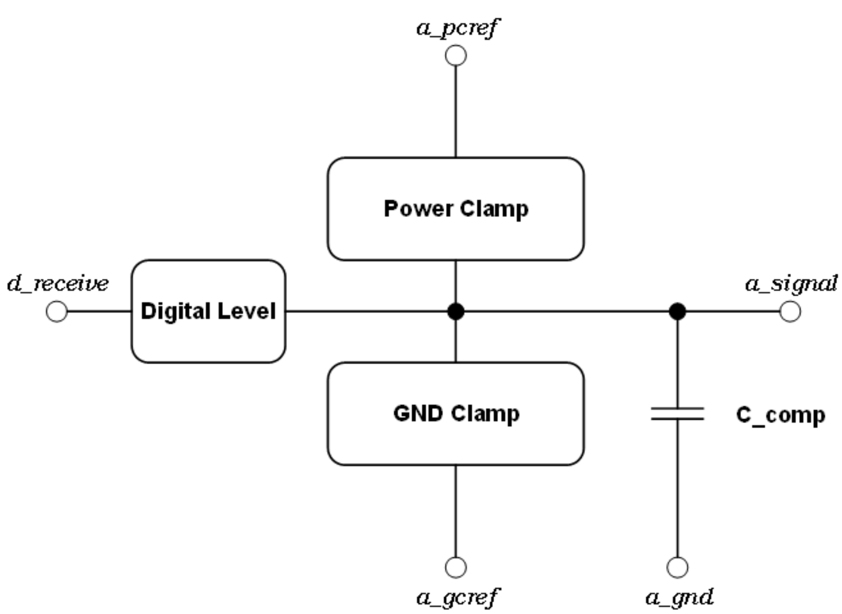
Type	Number	Nodes (in order)	Min/Max
Output_diff	19	a_signal_pos - a_signal_neg - d_drive - [PN]	3/7
I/O_diff	20	a_signal_pos - a_signal_neg - d_drive - d_enable - [d_receive] [PN]	4/9
3-state_diff	21	a_signal_pos - a_signal_neg - d_drive - d_enable - [PN]	4/8

Single-Ended Buffers

Input and Input_ECL buffers

Figure 23-1 shows the model of an Input buffer (receiver). It has two sets of I-V curves, a ground clamp and a power clamp, and the die capacitance *C_comp*. Two thresholds are defined, *Vinl* and *Vinh*, that determine the buffer's digital output, *d_receive*.

Figure 23-1. Input Buffer Model Building Blocks



The following relation gives the digital output for the non-inverting buffers:

Vd_receive =

1.0 V if *Va_signal* > *Vinh*

0.5 V if *Vinl* < *Va_signal* < *Vinh*

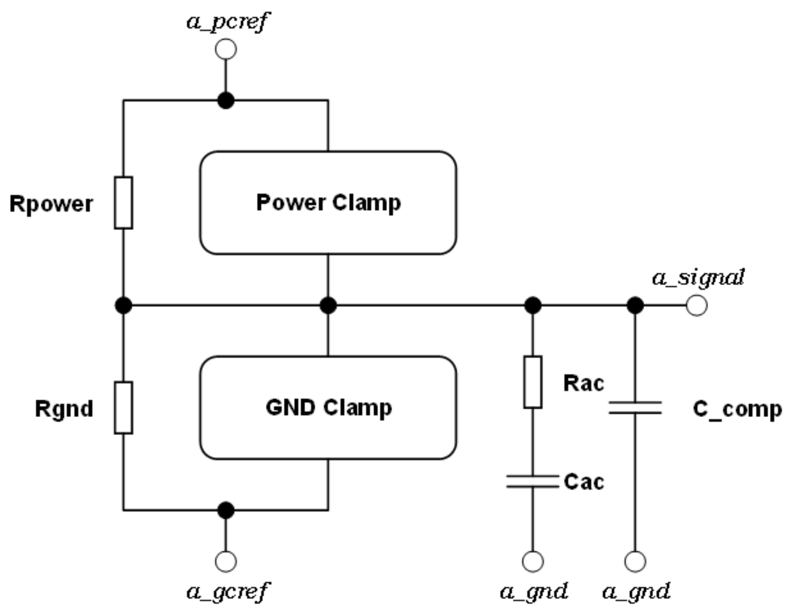
0.0 V if *Va_signal* < *Vinh*

If *Vinl* and *Vinh* are not specified in the IBIS file, the default values of *Vinl* = 0.8 V and *Vinh* = 2.0 V are assumed for an Input buffer and *Vinl* = -1.475 V and *Vinh* = -1.165 V are assumed for an Input_ECL buffer.

Terminator Buffers

A Terminator is an input-only model that can have analog loading effects on the circuit being simulated, but has no digital output. The Terminator model is shown in Figure 23-2. It may contain termination elements like *Rgnd*, *Rpower*, *Rac*, and *Cac* in addition to the usual clamping diodes circuitry.

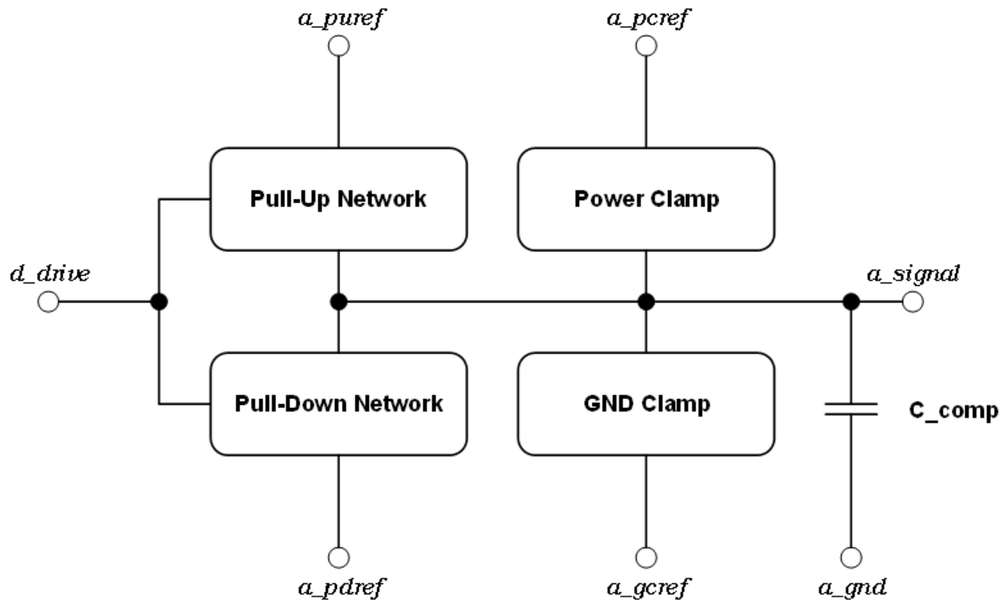
Figure 23-2. Terminator Buffer Model Building Blocks



Output and Output_ECL buffers

Figure 23-3 shows the model of an Output buffer (driver). It has four sets of I-V curves, a ground clamp, a power clamp, a pull-up network and a pull-down network, and the die capacitance *C_comp*. In addition, the transient switching characteristics of the pull-up and pull-down networks are also defined.

Figure 23-3. Output Buffer Model Building Blocks



IBIS provides two ways for defining the switching characteristics:

- If the output switching (V-T) waveform of a buffer can be approximated by a linear ramp then the V-T data may be reported as a rising and falling ramp rate (dV/dt) by using the [Ramp] keyword.
- Using the [Rising Waveform] and [Falling Waveform] keywords if the output switching waveform of the buffer is significantly non-linear.

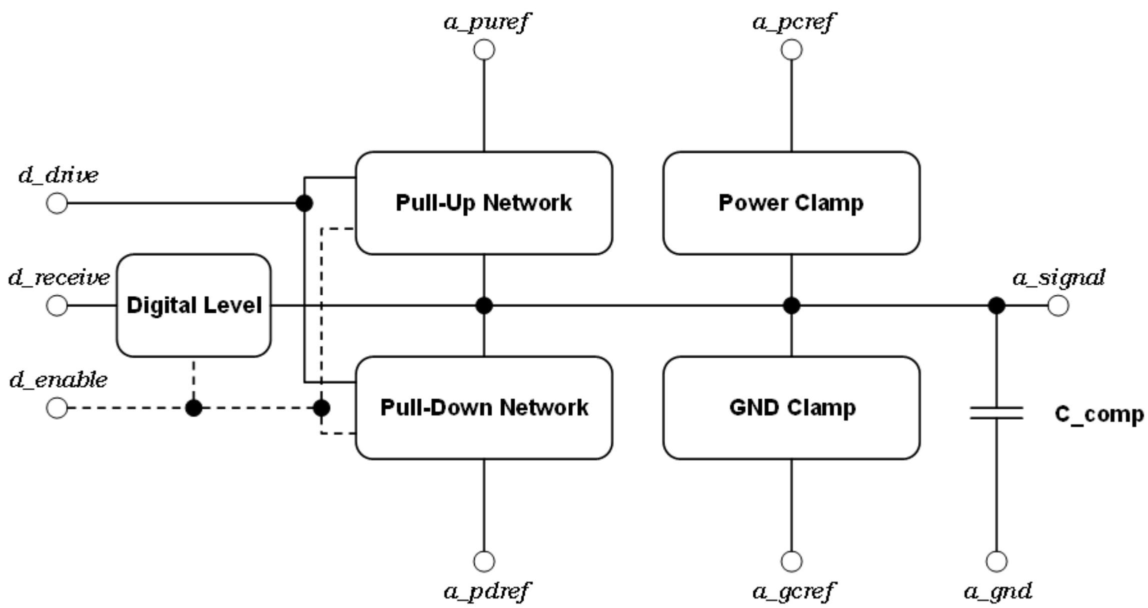
In both cases the transient current is considered to be a factor from the DC (steady state) current and this factor is calculated from the provided switching data. There are two factors; *k_pullup* and *k_pulldown*. These factors range from 0, representing a switched off pull-up/pull-down network, to 1, representing a fully switched on network. The switching starts when *d_drive* crosses 0.5 V and according to the buffer polarity, inverting or non-inverting, the appropriate network is switched on or off.

Output_ECL buffer differs from Output buffer in that the *a_pdref* is internally connected to the *a_puref*; that is, pull-up and pull-down share the same power reference. Output and Output_ECL buffers also differ in the conventions related to the [Pulldown], [Temperature Range], [Pin Mapping] keywords and the measuring conditions of the switching characteristics.

I/O and I/O_ECL Buffers

Figure 23-4 shows the model of an I/O buffer. The *d_enable* signal determines if the buffer will operate as an Input or Output buffer. If the buffer is active low, then it will behave as an Output buffer if *d_enable* < 0.5 V and as an Input buffer otherwise. If the buffer is active high, then it will behave as an Output buffer if *d_enable* > 0.5 V and as an Input buffer otherwise.

Figure 23-4. I/O Buffer Model Building Blocks

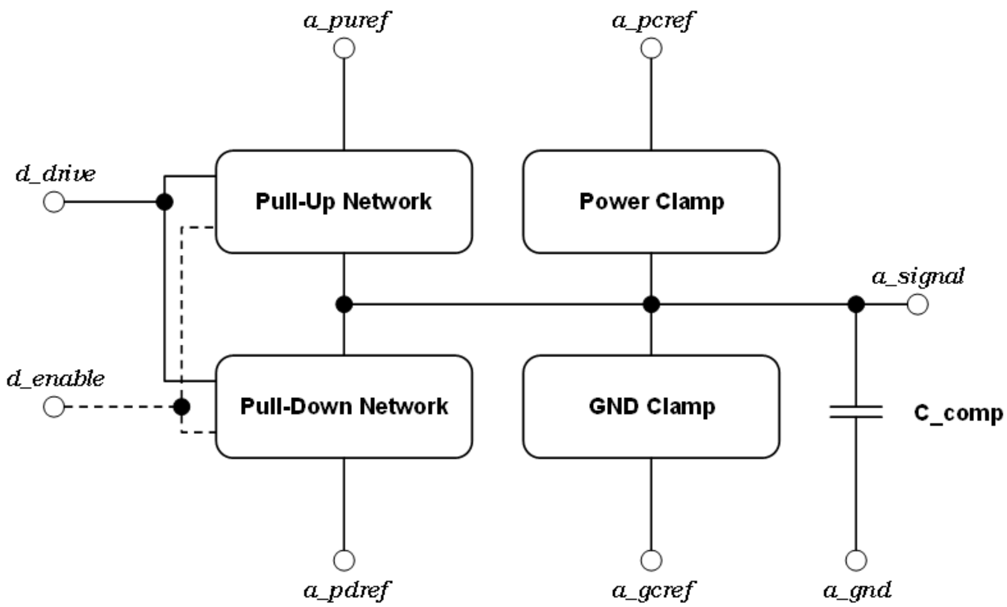


When behaving as an Output buffer, I/O_ECL buffer differs mainly from I/O buffer in that the *a_pdref* is internally connected to the *a_puref*; that is, pull-up and pull-down share the same power reference. I/O and I/O_ECL buffers also differ in the conventions related to [Pulldown], [Temperature Range], [Pin Mapping] keywords and the measuring conditions of the switching characteristics. Otherwise, if behaving as an Input buffer, I/O and I/O_ECL buffers differ in the default values of *Vinl* and *Vinh* (see "Input and Input_ECL buffers" on page 1290 for details).

3_state and 3_state_ECL Buffers

The model of a 3_state buffer is show in Figure 23-5. The 3_state buffer is very similar to the I/O buffer but does not have a digital output. It either works as an output buffer when enabled or high impedance when not enabled. The high impedance state is described by the Power Clamp, GND Clamp and die capacitance, *C_comp*.

Figure 23-5. 3_state Buffer Model Building Blocks



3_state_ECL buffer differs mainly from 3_state buffer in that the *a_pdref* is internally connected to the *a_puref*; that is, pull-up and pull-down share the same power reference. 3_state and 3_state_ECL buffers also differ in the conventions related to [Pulldown], [Temperature Range], [Pin Mapping] keywords and the measuring conditions of the switching characteristics.

Buffers with Open Drain, Sink, or Source

The open drain and open sink are buffers that do not include a pull-up network; that is, the output can sink current only. Earlier IBIS versions used the term "open drain", however, this may be confusing for NMOS and PMOS networks, and so the term "open sink" is now used to describe a buffer that can sink current only. The "open drain" terminology is retained for backward compatibility.

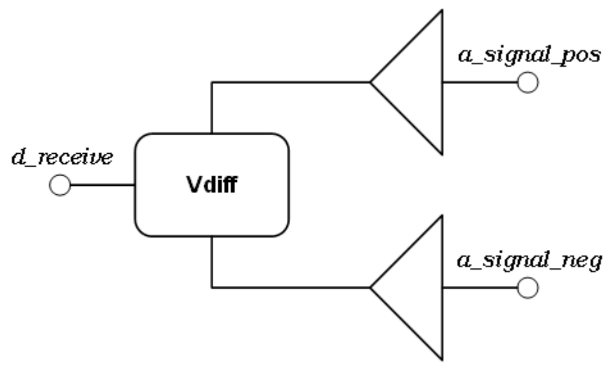
The open source buffer is a buffer that does not include a pull-down network; that is, the output can source current only.

Pseudo-Differential Buffers

The [Diff Pin] keyword in the IBIS file is used to create a pseudo-differential buffer from two already existing single-ended buffers.

A pseudo-differential input buffer consists of two single-ended input buffers as shown in Figure 23-6. The digital outputs of these two single-ended buffers are tied together and labeled by *d_receive*. One single-ended buffer's input is taken to be the non-inverting input (*a_signal_pos*) while the second one is taken to be the inverting input (*a_signal_neg*). The differential input threshold is denoted by *Vdiff* and is provided under the [Diff Pin] keyword in the IBIS file.

Figure 23-6. A Pseudo-Differential Input Buffer Consisting of two Single-Ended Input Buffers



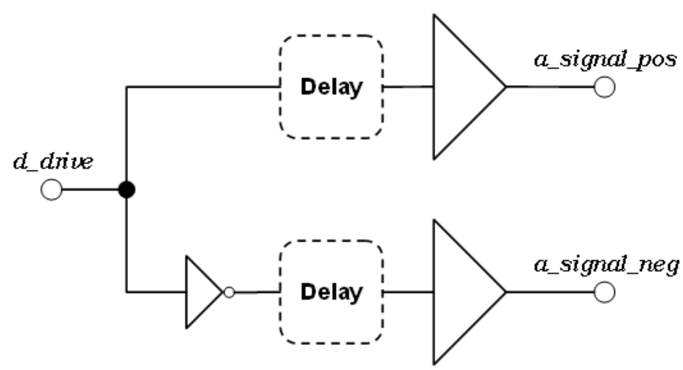
The digital output *d_receive* is given by:

d_receive =

- 1 V if *a_signal_pos* - *a_signal_neg* > V_{diff}
- 0 V if *a_signal_pos* - *a_signal_neg* < V_{diff}

The pseudo-differential output buffer consists of two single-ended output buffers as shown in Figure 23-7. One single-ended buffer's output is taken to be the non-inverting output (*a_signal_pos*) while the second one is taken to be the inverting output (*a_signal_neg*). The differential delay between the two outputs is denoted by *tdelay* and provided under the [Diff Pin] keyword in the IBIS file.

Figure 23-7. A Pseudo-Differential Output Buffer Consisting of two Single-Ended Output Buffers



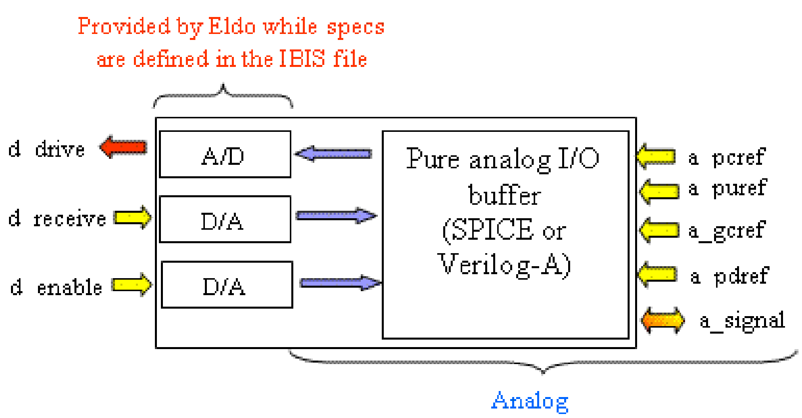
The two inputs are tied together and the overall differential buffer has one input and two outputs. Positive differential delay means the non-inverting output is delayed with respect to the inverting output. Inverting differential delay means inverting output is delayed with respect to the non-inverting output.

True Differential Buffers

The native IBIS does not support true differential buffers, they are only supported using the external model format created using SPICE or Verilog-A.

SPICE or Verilog-A formatted models are pure analog. Eldo is responsible for handling the A/D and D/A conversions as shown in Figure 23-8. Specification of these A/D and D/A blocks are defined in the IBIS file.

Figure 23-8. An Analog-Only Model init using an I/O Buffer as an Example



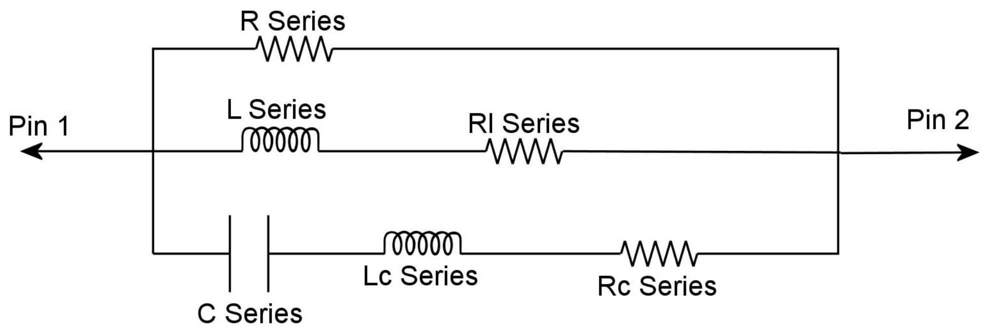
Series Buffers

Series and Series_switch buffers can be passive circuits containing various combinations of inductors, resistors and capacitors, as well as the steady state I-V characteristics of non-linear devices such as diodes and transistors. Series and Series_switch are identical in their elements, the only difference is that the latter one can be switched ON or OFF. The IBIS specifications do not define the transition characteristics of a Series_switch. Switches are assumed to either be ON or OFF during a simulation, and I-V characteristics could be defined for either or both states. Switching does not occur during simulation, but you must decide whether the element is ON or OFF before the simulation starts.

The electrical models for the Series / Series_switch are:

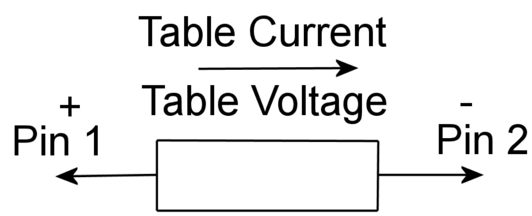
- R Series, L Series, Rl Series, C Series, Lc Series and Rc Series
  These enable IBIS to model simple passive models and/or parasitics. The model is shown in Figure 23-9.

Figure 23-9. Simple Passive Modeling



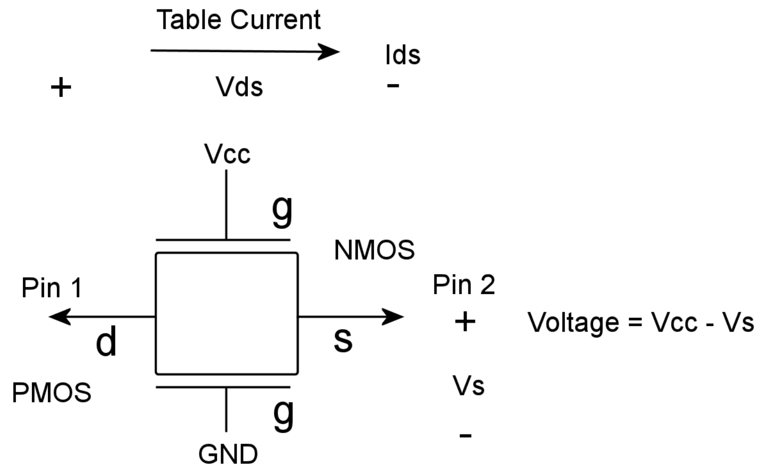
- Series Current
  under which the data points define the I-V tables for voltages measured at Pin 1 with respect to Pin 2. The model is shown in Figure 23-10. Currents are considered positive if they flow into Pin 1.

Figure 23-10. Series Current Modeling



- Series MOSFET
  under which the data points define the I-V tables for voltages measured at Pin 2 for a given Vds setting. Currents are considered positive if they flow into Pin 1. The model is shown in Figure 23-11.

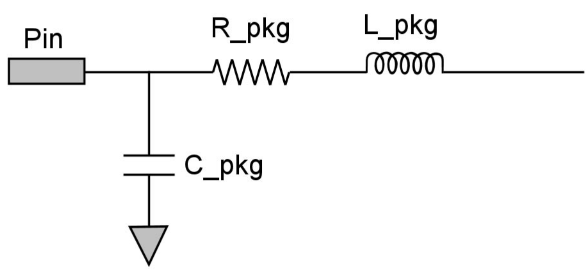
Figure 23-11. Series MOSFET Modeling



Package

IBIS has many ways for package modeling, differing in their complexity and accuracy. The simplest, least accurate, way is to assume all the package pins are identical and uncoupled. The common package parasitics, *R_pkg*, *L_pkg* and *C_pkg* shown in Figure 23-12, are defined under the [Package] keyword in the IBIS file. Also unique parasitics *R_pin*, *L_pin* and *C_pin* can be defined separately for each pin under the [Pin] keyword.

Figure 23-12. Each Pin is Assumed to have Parasitic Resistance R_pkg, Inductance L_pkg, and Capacitance C_pkg



More advanced package modeling can be achieved using the [Package Model] keyword, where you can refer to a package model inside the *.ibis* file or in a *.pkg* file. You can model the package as coupled pins and give the resistance, inductance and capacitance matrices (RLC matrices) elements of the package pins network. These matrices assume the Maxwellian format.

Another way of modeling is to divide the package into stub sections and define the electrical parameters for each section. The stub represents the physical connection between the package pin and the die pad. Each section is described in terms of its L/R/C per unit length and the Fork and Endfork subparameters allow any path to branch. For example, this can describe the parasitics of the wire bond between the die pad and the package pin. This stub can be treated as lumped or distributed elements depending on whether the section length is 0 or not.

Component

As mentioned earlier, Eldo can simulate IBIS devices either individually or in the overall component. The component simulation means creating all the buffers inside the component, connecting these buffers to the appropriate nodes and power buses, creating the package parasitics and connecting buffer die pads to the package parasitics. Several keywords and sub-parameters in the IBIS file are used in a component simulation in addition to that used in the individual buffer simulation:

- [Pin Mapping] is used to define the buffers sharing the same power/ground bus.
- [Series Pin Mapping] is used to join two die pads by a series buffer and [Series Switch Groups] is used to define switching combinations of series switches.

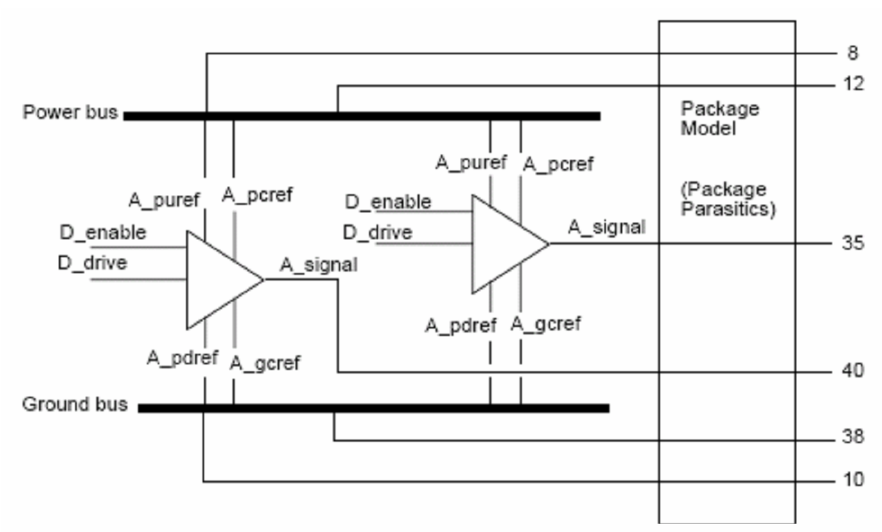
- [Circuit Call] and [Node Declarations] create subcircuits from external circuit models written in SPICE or Verilog-A language, and make their interconnections.
- [Package Model] defines an advanced package model, taking coupling effects into account.

Pin Mapping

When the input of a buffer(s) changes from High to Low, the output driver current (Ldi/dt) creates a fluctuation between the core ground and power buses. This creates a pulse that can affect static buffer(s) output, causing the receiver(s) to switch inappropriately. A similar effect occurs when the input of a buffer(s) changes from Low to High. The difference between the core ground and power is referred to as “ground bounce.” The amount of ground bounce is dependent on the number of outputs changing state at the same time and thus the ground bounce may also be called “simultaneous switching output noise” (SSON).

The [Pin Mapping] keyword, in the IBIS file, contains information on how power supplies are connected to individual buffers or groups of buffers that can be used for predicting SSON. The bus connections (from buffer nodes to supply or ground nodes) described by [Pin Mapping] are assumed as ideal shorts, and do not override parasitic information given for power and/or ground pins. Figure 23-13 indicates how the model is constructed. For example, pins 8 and 12 are connected to the Power bus; pins 10 and 38 are connect to the Ground bus; and the models with signal terminals connected to pins 35 and 40 use these buses for their voltage supplies.

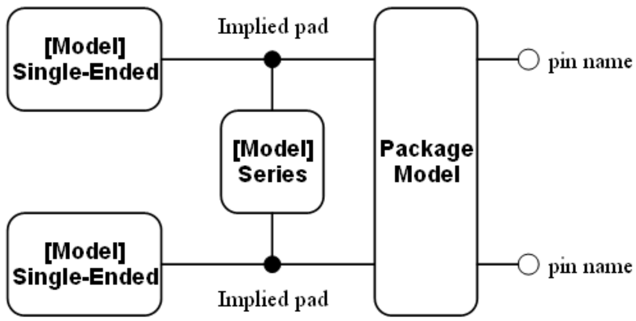
Figure 23-13. [Pin Mapping] Contains Information about Bus Connection from Buffer Nodes to Supply/ Ground Nodes



Series Pin Mapping and Series Switch Groups

[Series Pin Mapping] allows modeling of elements in series with a signal path, as shown in Figure 23-14. They are particularly useful for modeling elements placed between the terminals of differential buffers. However, the [Series Pin Mapping] keyword can generally be used to connect series elements between the die pads of any buffers inside the component.

Figure 23-14. Series Elements can be Connected Between Buffer Die Pads using the [Series Pin Mapping] Keyword



IBIS defines two categories of series elements; series and series switch. Series and series switch are identical in their elements. The only difference is that series switch can be switched ON or OFF. Series switches are divided into groups, and the group name to which the series switch belongs to, is defined in the function_table_group column in the IBIS file. The [Series Switch Groups] keyword is used to define switching combinations of series switch groups. You can specify the state of some groups to be ON and it is implicitly assumed that unspecified groups are OFF, or vice versa.

Node Declarations and Circuit Call

The [External Circuit] keyword allows you to define any block behavioral models, written in SPICE or Verilog-A, with any number of ports and with any functionality. The connectivity of these blocks to each other, to die nodes or die pads, must be defined using the [Node Declaration] and [Circuit Call] keywords. Only one [Node Declarations] keyword is permitted for each [Component] keyword. Multiple [Circuit Call] keywords may appear under a [Component] using the same [External Circuit] name if multiple instantiations of an [External Circuit] are needed.

Electrical Board Description

A "board level component" is a term describing a printed circuit board (PCB) or substrate which can contain components or even other boards, and which can connect to another board through a set of user visible pins. The electrical connectivity of such a board level component is referred to as an Electrical Board Description (EBD). An EBD file (*.ebd*) describes the connections of a board level component between the board pins and its components on the board. An *.ebd* file is intended to be a stand-alone file, not referenced by or included in any *.ibs* or *.pkg* files.

The IBIS EBD describes the connection between the user accessible pins of a board level component and other pins of the board or pins of the ICs mounted on that board. Each pin-to-node connection is divided into one or more cascaded sections, where each section is described in terms of its L/R/C per unit length. The Fork and Endfork subparameters allow any path to branch to multiple nodes, or another pin. A path description is required for each pin whose signal name is not GND, POWER or NC.

IBIS Support in Eldo

Eldo supports IBIS version IBISv4.2, this is the default. Releases prior to AMS 2007.2 supported IBISv2.1. The v4.2 support is a new implementation, not an extension to v2.1. You can select the old v2.1 solution, instead of the v4.2 one, using the Eldo option and IBIS instance keyword, **IEVER=1**.

Eldo can simulate IBIS devices either individually or in the overall component. See "Supported Keywords and Sub-Parameters" on page 1324 for further information.

Analysis Types

IBIS specifications provide data about the large signal behavior of the I/O buffers and their switching characteristics. However, no data about the frequency domain behavior is provided. Therefore, IBIS simulation can be performed in either the DC or Transient domain. IBIS simulation is supported in Eldo using:

- DC analysis
- Transient analysis

Digital Levels

IBIS defines both analog and digital ports. Table 23-3 and Table 23-4 show the different digital ports logic levels and the corresponding analog levels.

Table 23-3. Digital Port Logic Levels

Analog Input	Digital Input: d_drive, d_enable and d_switch
Vin > 0.5V	Logic "1"
Vin < 0.5V	Logic "0"

Table 23-4. Analog Levels

Digital Output: d_receive	Analog Output
Logic "1”	1.0 V
Logic "0”	0.0 V
Logic "X”	0.5 V

IBIS Buffers

Syntax

You can reference an IBIS buffer by a model name, or a pin of component. The syntax of IBIS buffers is as follows:

Single-Ended Buffer

```
_IO_xx ASG [DDR] [DEN] [DRX] [PCR GCR PUR PDR] file="path"
+ model="model_name"|pin="pin_name" [component="component_name"]
+ [device=type_name|type_number] [corner=TYP|MIN|MAX|FAST|SLOW]
+ [warn=ON|OFF] [nowarn] [power=ON|OFF]
+ [c_comp_pu=value] [c_comp_pd=value]
+ [c_comp_pc=value] [c_comp_gc=value]
+ [use_fall_wvf=0|1|2] [use_rise_wvf=0|1|2]
+ [k_pulldown=node] [k_pullup=node]
+ [model_selector='msel_1=mdl_1 [,…, msel_n=mdl_n]'] [msel_mode=value]
+ [package=0|1|2|3|OFF|CMPNT|PIN|PKGMOD] [pkg_model="pkg_model_name"]
+ [rx_logic=0|1]
+ [para_begin param1=value1 param2=value2 ... para_end]
+ [eldova=ON|OFF] [em_libname="logical_lib_name"]
+ [table_tune=value]
+ [logic_one=value] [logic_zero=value]
+ [c_fixture=YES|NO|IGNORE]
```

Pseudo-Differential Buffer

```
_IO_xx ASP ASN [DDR] [DEN] [DRX] [PCR GCR PUR PDR] file="path"
+ component="component_name" pin="pin_name" inv_pin="inv_pin_name"
+ [corner=TYP|MIN|MAX|FAST|SLOW] [warn=ON|OFF] [nowarn] [power=ON|OFF]
+ [c_comp_pu=value] [c_comp_pd=value]
+ [c_comp_pc=value] [c_comp_gc=value]
+ [use_fall_wvf=0|1|2] [use_rise_wvf=0|1|2]
+ [model_selector='msel_1=mdl_1 [,…, msel_n=mdl_n]'] [msel_mode=value]
+ [package=0|1|2|3|OFF|CMPNT|PIN|PKGMOD] [pkg_model="pkg_model_name"]
+ [para_begin param1=value1 param2=value2 ... para_end]
+ [eldova=ON|OFF] [em_libname="logical_lib_name"]
+ [table_tune=value]
+ [logic_one=value] [logic_zero=value]
+ [add_series_terminator=YES|NO|IFA]
+ [c_fixture=YES|NO|IGNORE]
```

True Differential Buffer

```
_IO_xx ASP ASN [DDR] [DEN] [DRX] [PCR GCR PUR PDR] file="path"
+ model="model_name"|pin="pin_name" [inv_pin="inv_pin_name"]
+ [component="component_name"]
+ [device=type_name|type_number] [corner=TYP|MIN|MAX|FAST|SLOW]
+ [warn=ON|OFF] [nowarn] [power=ON|OFF]
+ [model_selector='msel_1=mdl_1 [,…, msel_n=mdl_n]'] [msel_mode=value]
+ [package=0|1|2|3|OFF|CMPNT|PIN|PKGMOD] [pkg_model="pkg_model_name"]
+ [para_begin param1=value1 param2=value2 ... para_end]
+ [eldova=ON|OFF] [em_libname="logical_lib_name"]
+ [table_tune=value]
+ [add_series_terminator=YES|NO|IFA]
```

Series Buffer

```
_IO_xx APV ANV [DST] file="path"
+ model="model_name"
+ [device=type_name|type_number][corner=TYP|MIN|MAX|FAST|SLOW]
+ [warn=ON|OFF] [nowarn]
+ [ss_state=ON|OFF] [all_sm=YES|NO]
+ [keep_vds_monotonic=NO|YES] [keep_vgs_monotonic=NO|YES]
+ [model_selector='msel_1=mdl_1 [,…, msel_n=mdl_n]'] [msel_mode=value]
+ [para_begin param1=value1 param2=value2 ... para_end]
+ [eldova=ON|OFF] [em_libname="logical_lib_name"]
```

Parameters

Some parameters are part of the IBIS standard, please refer to description in "IBIS Device Standards" on page 1288.

- **_IO_xx**

  IBIS instance name.

- ASG

  Name of the a_signal node.

- ASP

  Name of the a_signal_pos node.

- ASN

  Name of the a_signal_neg node.

- DDR

  Name of the d_drive node.

- DEN

  Name of the d_enable node.

- DRX

  Name of the d_receive node.

- APC

  Name of the a_pcref node.
- AGC

  Name of the a_gcref node.
- APU

  Name of the a_puref node.
- APD

  Name of the a_pdref node.
- APV

  Name of the a_pos node.
- ANG

  Name of the a_neg node.
- DST

  Name of d_switch node, this node is used only for Series_Switch buffers that reference an external model.

Note

Buffer external nodes should be listed in order, depending on the type of the buffer model. A list of the buffer model types and corresponding ports are in Table 23-2 on page 1289.

- **file=**"path"

  Specifies the path to the file that contains an IBIS formatted model.

  This can be the full path (for example, */user/test/Models/model.ibs*), a relative path, or just a file name. Both / and \ are acceptable as separators in the path name for the UNIX and Windows platforms. An Eldo option IBIS_SEARCH_PATH is available to specify the path to the directory to search for the IBIS files.
- **model=**"model_name"

  Specifies an IBIS model name within the specified IBIS file, it can be either a model name or a model selector name. Model name is case-sensitive.
- **pin=**"pin_name"

  Specifies a pin name within the specified component.

Note

The keywords **pin** and **model** are exclusive. It is an error to specify both these keywords in the same **_IO_ card**.

- **inv_pin**="inverting_pin_name"

  Specifies a pin name within the specified component; this is to be used with the pseudo/true differential buffers. Pin name and inverting pin name must match a differential pins pair under [Diff Pin] IBIS keyword.

- **component**="component_name"

  Specifies a component name within the specified IBIS file. The component name is case-sensitive. Component name must be specified if pin name or inverting pin name is specified, while it is optional if you specify a model name.

- **device**="type_name|type_number"

  Specifies the desired buffer type to be used; this is to check the buffer type assignment in the IBIS instance against the buffer type given in the IBIS file. A list with buffer types and numbers are given in Table 23-2 on page 1289.

- **corner**=TYP|MIN|MAX|FAST|SLOW

  Specifies the IBIS corner to be used to extract the data from the IBIS file. The default is **corner**=TYP—if any data is missing under the Min or Max columns in the IBIS file then the Typ column value will be used instead. When **corner**=TYP, MIN, or MAX, simulation will extract the data under the corresponding Typ, Min, and Max columns in the IBIS file. When **corner**=FAST or SLOW, corner uses combinations from data under the Min and Max columns; FAST is the same as MAX, and SLOW is the same as MIN except for parameters listed in Table 23-5.

Table 23-5. Exceptions to Fast=Max, Slow=Min Rule

Parameter/Data	Fast	Slow
C_comp	Min	Max
[Cac]	Min	Max
[Gnd Clamp Reference]	Min	Max
[Pulldown Reference]	Min	Max
[Package]	Min	Max

- **power**=ON|OFF

  When **power**=ON (the default), the reference voltage sources are connected to the buffer reference ports *a_pcref*, *a_gcref*, *a_puref*, *a_pdref* internally. When **power**=OFF you are responsible for connecting the voltage sources externally, if you set **power**=OFF but do not specify the power nodes in the **_IO_ card**, then OFF will be ignored and internal references will be created.

  Power nodes can be specified when **power**=ON to allow you to print/plot the values of voltage sources connected to these nodes internally, in this case you must not connect external voltage sources to these nodes.

- **warn=**ON|OFF

  Specifies if the warning message generated for this instance should be printed: ON, or suppressed: OFF. Setting **warn=**OFF is equivalent to using the keyword **nowarn**. The default setting is ON.

- **nowarn**

  This is equivalent to **warn=**OFF.

- **c_comp_pu=**value, **c_comp_pd=**value, **c_comp_pc=**value, **c_comp_gc=**value

  These four keywords are optional. If one of these keywords is specified with a non-zero value; the *c_comp* capacitor will be split up to four parts. Values are dimensionless numbers between 0 and 1, and the sum of them should be equal to 1. The default is zero for the unspecified values.

- **use_fall_wvf=** 0|1|2, **use_rise_wvf=** 0|1|2

  0

  Use the ramp for the falling/rising transitions.

  1

  Use one waveforms for the falling/rising transitions.

  2

  Use two waveforms for the falling/rising transitions.

  The default behavior is to use the first two waveforms if more than one waveform is specified in the IBIS model, use one waveform if only one is specified, and to use ramp if no waveforms are specified.

- **k_pulldown=**"node", **k_pullup=**"node"

  The purpose of these keywords is to display the values of the pull-up and pull-down coefficients of the device as a function of time as the device goes through rising or falling transitions.

- **msel_mode=**value

  The value indicates a model order under the [Model Selector] keyword in the IBIS file. If the value given is greater than the model selector list, then the first model will be selected. The default is 1.

- **model_selector=**'mod_sel1=mod_name1 [, mod_sel2=mod_name2 ,...]'

  Used to select models for some or all model selectors in the given IBIS file. **model_selector** has a higher precedence over **msel_mode**.

- **package=**0|1|2|3|OFF|CMPNT|PIN|PKGMOD

  Specifies the package modeling type to be used for package parasitics.

  0 or OFF

  No package is added.

1 or CMPNT

RLC components defined under [Package] keyword are added, this is the default when specifying model name.

2 or PIN

RLC components defined under [Pin] are added, this is the default when specifying pin name.

3 or PKGMOD

A package model will be used. This is the default.

Note

If no data is available for selected packaging type, then the data in the lower type will be used.

- **ss_state**=ON|OFF

  For Series Switch buffers, if **ss_state=**ON, then Eldo will use the data under the [On] keyword: setting it to OFF, the data under [Off] keyword will be used. The default is ON.

- **all_sm=**YES|NO

  For Series/Series_Switch buffers; setting **all_sm** to NO makes Eldo use only the first non-zero Vds table to generate the current of Series MOSFET elements in the Series buffer. The default is YES.

- **keep_vds_monotonic=**NO|YES

  For Series/Series_Switch buffers; setting **keep_vds_monotonic** to YES will keep the Ids current monotonic versus Vds values for a given Vgs. Default is NO.

- **keep_vgs_monotonic**=NO|YES

  For Series/Series_Switch buffers; setting **keep_vgs_monotonic** to YES will keep the Ids current monotonic versus Vgs values for a given Vds. Default is NO.

- **rx_logic**=0|1

  For Input or IO buffers; if the input voltage on a_signal port is between Vinl and Vinh thresholds then **rx_logic** will control the output voltage on the d_receive port. When **rx_logic**=0 (default), the voltage on the d_receive port will be 0.5V (representing logic "X"). When **rx_logic**=1, the voltage on the d_receive port will have the same value (state) as the previous time step.

Note

If there are hysterises thresholds (Vinl+, Vinl-, Vinh+, and Vinh-) under the [Model Spec] keyword, then **rx_logic** has no effect.

- **para_begin** param1=value1 param2=value2 ... **para_end**

  For IBIS buffers that reference an external model, the parameter names and values specified between keywords para_begin/para_end will be passed to the external model module.

- **eldova=**ON|OFF

  For IBIS models that reference an external model in Verilog-A, Eldo will use its Verilog-A compiler to handle these models if **eldova**=ON. Setting **eldova**=OFF enables these models to be simulated with Questa ADMS. The default is ON.

- **em_libname=**"logical_lib_name"

  For IBIS models that reference external models written in any AMS language, this keyword is optionally used. If no library is specified, the default work library will be used, specified with the **vasetlib** command or the **-lib** option of the **vasim** command. Otherwise, the specified logical_lib_name library is used.

- **table_tune=**value

  Controls Eldo time step when the IBIS buffer is switching to obtain accurate output waveforms. Table_tune accepts non-negative values. Default value is 8.

  Setting a value of 1 forces Eldo to take into account all the switching waveform points, 2 to take one point into account and skip the next, 3 skips two points, but takes into account the third, and so on. Setting a value of 0 makes Eldo control the time step regardless of the switching waveform.

Note

The actual time step will be the minimum among the time step of the current IBIS buffer, time steps of other IBIS buffers, and the time step forced by Eldo.
HMIN is always set as the lower limit for the time step.

- **logic_one=**value|**logic_zero=**value

  Define the voltage level accepted by IBIS buffers for digital ports, **logic_one** default is 1.0, and **logic_zero** default is 0.0. **logic_one** must be greater than **logic_zero** by at least 200mV.

  The input voltage is considered ‘logic 1’ if it is greater than (**logic_one** + **logic_zero**)/2, and considered ‘logic 0’ if it is below this value.

Note

These levels are defined for native IBIS buffers and do not apply to external models.

- **add_series_terminator=**YES|NO|IFA

  Used with the pseudo/true differential buffers to enable/disable creation of series models across the differential buffers die pads. Default is NO. IFA is the same as YES but does not produce warning messages if there is no series models defined under [Series Pin Mapping] for the differential buffers pins.

- **c_fixture**=YES|NO|IGNORE

  Enable/disable using rising or falling waveforms that have a c_fixture sub-parameter with a non-zero value. Setting to NO (default) makes Eldo discard these waveforms; YES to use these waveforms; and IGNORE to use these waveforms but with setting c_fixture value to zero (ignore c_fixture).

Example

```
IBIS File Example

[IBIS Ver]       3.2
[Comment Char]   |_char
[File Name]      dummy.ibs
|
   .
   .
   .
|
[Component]      Test_Component
[Manufacturer]   None
[Package]
|                 typ        min          max
R_pkg            0.020      0.017        0.025
L_pkg            0.998nH    0.881nH      1.069nH
C_pkg            0.146pF    0.118pF      0.205pF
|
|**********************************************************
|
[Pin]  signal_name  model_name  R_pin   L_pin   C_pin
|
1      NC           NC          0.019   1.069n  0.162p
2      in           MS_In       0.018   0.881n  0.121p
3      out+         MS_Out      0.017   1.031n  0.205p
4      out-         MS_Out      0.017   1.031n  0.205p
|
[Model Selector]  MS_In
Input_33          Vcc = 3.3 V
Input_50          Vcc = 5.0 V
|
[Model Selector]  MS_Out
Output_33         Vcc = 3.3 V
Output_50         Vcc = 5.0 V
|
[Diff Pin] inv_pin  vdiff  tdelay_typ  tdelay_min  tdelay_max
|
3          4        150mV  -1ns        0ns         -2ns
|
[Series Pin Mapping] pin_2 model_name
|
3                    4     Series1
|
|**********************************************************
|                       Model Input_33
|**********************************************************
|
[Model]     Input_33
Model_type  Input
Vinl = 0.99
Vinh = 2.31
|
```

```
| variable   typ      min      max
C_comp       1.27p    0.95p    1.59p
|
[Temperature Range]      25      85      -40
[Voltage Range]          3.3     3.0     3.6
|
   .
   .
   .
|
|**********************************************************
|                       Model Input_50
|**********************************************************
|
[Model]     Input_50
Model_type  Input
Vinl = 1.2
Vinh = 3.5
|
| variable   typ      min      max
C_comp       1.27p    0.95p    1.59p
|
[Temperature Range]      25      85      -40
[Voltage Range]          5.0     4.5     5.5
|
   .
   .
   .
|
|**********************************************************
|                       Model Output_33
|**********************************************************
|
[Model]     Output_33
Model_type  Output
|
| variable   typ      min      max
C_comp       1.27p    0.95p    1.59p
|
[Temperature Range]      25      85      -40
[Voltage Range]          3.3     3.0     3.6
|
   .
   .
   .
|
|**********************************************************
|                       Model Output_50
|**********************************************************
|
[Model]     Output_50
Model_type  Output
|
| variable   typ      min      max
C_comp       1.27p    0.95p    1.59p
|
[Temperature Range]      25      85      -40
[Voltage Range]          5.0     4.5     5.5
```

```
|
   .
   .
   .
|
|**********************************************************
|                       Model Series1
|**********************************************************
|
[Model]     Series1
Model_type  Series1
|
| variable  R(typ) R(min) R(max)
[R Series]  8ohm   6ohm   12ohm
|
| variable  L(typ) L(min) L(max)
[L Series]  5nH    NA     NA
| variable  R(typ) R(min) R(max)
[Rl Series] 4ohm   NA     NA
|
   .
   .
   .
|
[End]
```

The preceding IBIS file example is referenced in the following netlist examples.

- Netlist 1

  ```
  _IO_input1_bymodel IN D_OUT1
  + file="dummy.ibs" model="Input_33"
  + device=input ! or device=1
  ```

  The above netlist calls the buffer by its model name. It instantiates an input buffer input1_bymodel and connects its analog input to node IN and its digital output to node D_OUT1. The buffer description is in the IBIS file *dummy.ibs* located in the same directory as the netlist. The buffer model, named Input_33, is picked up from the IBIS file.

  Note that the component name is not specified; therefore Eldo will not add any package parasitics to the buffer analog input pin. The corner is not specified, therefore the typical data will be used for this buffer by default. The power keyword is not specified, therefore Eldo connects the reference voltage sources to the buffer reference ports.

- Netlist 2

  ```
  _IO_input1_bypin IN D_OUT2 PC GC
  + file="dummy.ibs"
  + component="Test_Component" pin="2"
  + model_selector='MS_In=Input_50' ! equal to msel_mode=2
  ```

  The above netlist calls the buffer by its pin name. It instantiates an input buffer input1_bypin and connects its analog input to node IN and its digital output to node D_OUT2. The buffer description is in the IBIS file *dummy.ibs* located in the same directory as the netlist. It belongs to the component Test_Component. The buffer model, named

MS_In (the one associated with pin 2), is picked up from the IBIS file. The MS_In model has a model selector statement in the IBIS file. The netlist selects the Input_50 model for MS_In.

The package keyword takes the default of 2 because the buffer is instantiated by pin name. Therefore Eldo will add the package parasitics associated with pin 2 under the [Pin] keyword, R=0.018 L=0.881n C=0.121p, to the buffer analog input pin. The corner is not specified, therefore the typical data will be used for this buffer by default. The power keyword is not specified, therefore Eldo connects the reference voltage sources to the buffer reference ports. In this case the nodes PC and GC only have the role of enabling you to plot/print the voltage reference values.

- Netlist 3

```
* make pseudo-differential buffer for pin 3 and pin 4
_IO_out_diff OUTp OUTn d_CTRL
+ file="dummy.ibs" component="Test_Component"
+ pin="3" inv_pin="4"
```

The above netlist instantiates the pseudo-differential buffer associated with pins 3 and 4. The non-inverting pin of the buffer is connected to node OUTp, inverting pin to node OUTn, and the digital input to node d_CTRL. The buffer description is in the IBIS file *dummy.ibs* located in the same directory as the netlist. It belongs to the component Test_Component.

The package keyword takes the default of 2 because the buffer is instantiated by pin name. Therefore Eldo will add the package parasitics associated with pin 3 and 4 under the [Pin] keyword, R=0.017 L=1.031n C=0.205p, to the buffer analog input pins. The corner is not specified, therefore the typical data will be used for this buffer by default. The power keyword is not specified, therefore Eldo connects the reference voltage sources to the buffer reference ports.

- Netlist 4

```
*connect series buffer between 3 and 4
_IO_ser OUTp OUTn
+ file="dummy.ibs" model="Series1"
```

The above netlist instantiates the series buffer associated with pins 3 and 4. It is connected to nodes OUTp and OUTn. The buffer description is in the IBIS file *dummy.ibs* located in the same directory of the netlist. The buffer model is Series1. The corner is not specified, therefore the typical data will be used for this buffer by default.

Note that any series buffer has no package associated with it.

IBIS Component

Syntax

```
_IO_xx file="path" component="component_name"
+ [corner=TYP|MIN|MAX|FAST|SLOW] [warn=ON|OFF] [nowarn]
+ [model_selector='msel_1=mdl_1 [,…, msel_n=mdl_n]'] [msel_mode=value]
+ [package=0|1|2|3|4|OFF|CMPNT|PIN|PKGMOD|MIX]
+ [pkg_model="pkg_model_name"]
+ [ss_group='ON|OFF G1 [,…, Gn]'] [nopseudo]
+ [use_fall_wvf=0|1|2] [use_rise_wvf=0|1|2]
+ [c_comp_pu=value] [c_comp_pd=value] [c_comp_pc=value]
+ [c_comp_gc=value] [all_sm=YES|NO]
+ [keep_vds_monotonic=NO|YES] [keep_vgs_monotonic=NO|YES]
+ [rx_logic=0|1]
+ [eldova=ON|OFF] [em_libname="logical_lib_name"]
+ [table_tune=value]
+ [logic_one=value] [logic_zero=value]
+ [c_fixture=YES|NO|IGNORE]
```

Parameters

Some parameters are part of the IBIS standard, please refer to description in “IBIS Device Standards” on page 1288.

- **_IO_xx**

  IBIS instance name.

- **file**="path"

  Specifies the path to the file that contains an IBIS formatted model.

  This can be the full path (for example, */user/test/Models/model.ibs*), a relative path, or just a file name. Both / and \ should be acceptable as separators in the path name (for the UNIX and NT versions). An Eldo option IBIS_SEARCH_PATH is available to specify the path to the directory to search for the IBIS files.

- **component=**"component_name"

  Specifies a component name within the specified IBIS file. The component name is case-sensitive.

- **corner**=TYP|MIN|MAX|FAST|SLOW

  Specifies the IBIS corner to be used to extract the data from the IBIS file. The default is **corner**=TYP—if any data is missing under the Min or Max columns in the IBIS file then the Typ column value will be used instead. When **corner**=TYP, MIN, or MAX, simulation will extract the data under the corresponding Typ, Min, and Max columns in the IBIS file. When **corner**=FAST or SLOW, corner uses combinations from data under the Min and Max columns; FAST is the same as MAX, and SLOW is the same as MIN except for parameters listed in Table 23-5 on page 1305.

- **warn=**ON|OFF

  Specifies if the warning message generated for this instance should be printed (ON), or suppressed (OFF). Setting **warn=**OFF is equivalent to using the keyword **nowarn**. The default setting is ON.

- **nowarn**

  This is equivalent to **warn=**OFF.

- **msel_mode=**value

  The value indicates a model order under the [Model Selector] keyword in the IBIS file. If the value specified is greater than the model selector list, then the first model will be selected. The default is 1.

- **model_selector=**'mod_sel1=mod_name1 [, mod_sel2=mod_name2 ,...]'

  This keyword can be repeated more than once. **model_selector** is used to select models for some or all model selectors in the given IBIS file. **model_selector** has a higher precedence over **msel_mode**.

- **package=**0|1|2|3|4|OFF|CMPNT|PIN|PKGMOD|MIX

  This keyword specifies the package modeling type to be used for package parasitics.

  0 or OFF

  No package is added.

  1 or CMPNT

  RLC components defined under [Package] keyword are added.

  2 or PIN

  RLC components defined under [Pin] are added.

  3 or PKGMOD

  A package model will be used. This is the default.

  4 or MIX

  A package model will be used, and RLC parasitics are added as defined under [Pin] section for pins that are not defined in the package model.

Note

If no data is available for selected packaging type, then the data in the lower type will be used.

- **pkg_model=**"package_model_name"

  Specifies an IBIS package model name within the specified IBIS file. The package model name must match one defined for this component under [Package Model] or [Alternate Package Models] keywords. If **pkg_model** is not specified, the one under [Package Model] will be used. “package_model_name” is case-sensitive.

- **ss_group=**'ON|OFF G1 [,G2, ...]'

  Sets the On/Off state for the series switch buffers created according the info under [Series Pin Mapping] keyword.

  You can set the logical state of this set of groups by type ON|OFF in the begining of the set, and then follow it by the groups' names that will take this state. The specified set of groups must match one of these under [Series Switch Groups] keyword or its complementary. If ss_group is not specified Eldo will take the first set of groups under [Series Switch Groups] keyword to set the logical state for the series switch buffers.

- **nopseudo**

  When specified Eldo does not generate pseudo-differential buffers for the pseudo-differential pins, and generates single-ended buffers instead.

  By default Eldo generates pseudo or true differential buffers as defined under [Diff Pin] keyword.

Note

The following keywords are used in the generated buffers if applicable.

- **c_comp_pu**=value, **c_comp_pd=**value, **c_comp_pc=**value, **c_comp_gc=**value

  These four keywords are optional. If one of these keywords is specified with a non-zero value; *c_comp* capacitor will be split up to four parts. Values are dimensionless numbers between 0 and 1, and sum of them should be equal to 1. The default is 0 for the unspecified values.

- **use_fall_wvf=** 0|1|2, **use_rise_wvf=** 0|1|2

  0

  Use the ramp for the falling/rising transitions.

  1

  Use one waveforms for the falling/rising transitions.

  2

  Use two waveforms for the falling/rising transitions.

  The default behavior is to use the first two waveforms if more than one waveform is specified in the IBIS model, use one waveform if only one is specified, and to use ramp if no waveforms are specified.

- **all_sm=**YES|NO

  For Series/Series_Switch buffers. Setting **all_sm** to NO makes Eldo use only the first non-zero Vds table to generate the current of Series MOSFET elements in the Series buffer. The default is YES.

- **keep_vds_monotonic**=NO|YES

  For Series/Series_Switch buffers. Setting **keep_vds_monotonic** to YES will keep the Ids current monotonic versus Vds values for a given Vgs. Default is NO.

- **keep_vgs_monotonic**=NO|YES

  For Series/Series_Switch buffers. Setting **keep_vgs_monotonic** to YES will keep the Ids current monotonic versus Vgs values for a given Vds. Default is NO.

- **rx_logic**=0|1

  For Input or IO buffers; if the input voltage on a_signal port is between Vinl and Vinh thresholds then **rx_logic** will control the output voltage on the d_receive port. When **rx_logic**=0 (default), the voltage on the d_receive port will be 0.5V (representing logic "X"). When **rx_logic**=1, the voltage on the d_receive port will have the same value (state) as the previous time step.

Note

If there are hysterises thresholds (Vinl+, Vinl-, Vinh+, and Vinh-) under the [Model Spec] keyword, then **rx_logic** has no effect.

- **eldova**=ON|OFF

  For external models and external circuits written in Verilog-A. Eldo will use its Verilog-A compiler to handle these models if **eldova**=ON. Setting **eldova**=OFF enables these models to be simulated with Questa ADMS. The default is ON.

- **em_libname=**"logical_lib_name"

  Can be used for external models and external circuits written in any AMS language. The specified logical_lib_name library is used as the work library. If no library is specified, the default work library will be used, specified with the ADMS vasetlib command or the vasim **-lib** command.

- **table_tune=**value

  Controls Eldo time step when the IBIS buffer is switching to obtain accurate output waveforms. Table_tune accepts non-negative values. Default value is 8.

  Setting a value of 1 forces Eldo to take into account all the switching waveform points, 2 to take one point into account and skip the next, 3 skips two points, but takes into account the third, and so on. Setting a value of 0 makes Eldo control the time step regardless of the switching waveform.

Note

The actual time step will be the minimum among the time step of the current IBIS buffer, time steps of other IBIS buffers, and the time step forced by Eldo.
HMIN is always set as the lower limit for the time step.

- **logic_one=**value|**logic_zero=**value

  Define the voltage level accepted by IBIS buffers for digital ports, **logic_one** default is 1.0, and **logic_zero** default is 0.0. **logic_one** must be greater than **logic_zero** by at least 200mV.

  The input voltage is considered ‘logic 1’ if it is greater than (**logic_one** + **logic_zero**)/2, and considered ‘logic 0’ if it is below this value.

Note

These levels are defined for native IBIS buffers and do not apply to external models.

- **c_fixture**=YES|NO|IGNORE

  Enable/disable using rising or falling waveforms that have a c_fixture sub-parameter with a non-zero value. Setting to NO (default) makes Eldo discard these waveforms; YES to use these waveforms; and IGNORE to use these waveforms but with setting c_fixture value to zero (ignore c_fixture).

Equivalent subcircuit

Eldo uses the data under the [Component] keyword to generate buffers, interconnections, package parasitics, and SPICE subcircuits or Verilog-A modules that represent external circuit calls. Each component pin that references a model name or model selector name will be associated with a buffer. The pins that are used as differential pairs under the [Diff Pin] keyword will be connected to differential buffers, the other pins will be connected to single-ended buffers.

Series/Series_Switch models will be created according to the data under the [Series Pin Mapping] keyword. Power interconnections are done according to the data under the [Pin Mapping] keyword if available; otherwise internal source references are created for each buffer. Package parasitics are added according to the package modeling type specified by the **package** keyword, and the data available about the package model.

External nodes naming rules

Component pins that reference anything other than “NC” reserved keywords as the model name will be added to the Eldo netlist as external nodes for the component equivalent subcircuit. If the pin references an “NC” reserved keyword, but is used by the package model or exists in the pin mapping of the series buffers, it will be added to the netlist also.

Additional nodes are also added to make access for buffers input, output, and control signals. These additional nodes should be connected to ground if they will not be used.

The naming rules for the buffers and external nodes are:

- Component pins:

  ```
  <instance_name>_<pin_name>
  ```

- Control nodes:

  ```
  <instance_name>_<buffer_name>_<reserved_node_name>
  ```

Buffer names are as follows:

- Single-Ended Buffer:

  ```
  <pin_name>
  ```

- Differential Buffer:

  ```
  <diff_pin_name>_<inv_pin_name>_diff
  ```

- Series Buffer:

  ```
  <pos_pin>_<neg_pin>_<model_name>_<function_group_name>_sers
  ```

Example

This example uses the same IBIS file as that used in the “IBIS File Example” on page 1309.

- Netlist

  ```
  _IO_comp1 file="dummy.ibs"
  + component="Test_component"
  ```

  ```
  .connect _IO_comp1_2_d_receive _IO_comp1_3_4_diff_d_drive
  ```

  The above netlist instantiates the entire component Test_component located in the IBIS file *dummy.ibs*. The component instance is given the name _IO_comp1. It is composed of an input buffer, a pseudo-differential buffer, and a series buffer (as shown in Figure 23-15). The digital output of the input buffer _IO_comp1_2_d_receive and the digital input of the differential output buffer _IO_comp1_3_4_diff_d_drive are connected in the netlist using the .connect statement.

Figure 23-15. Test_component Example

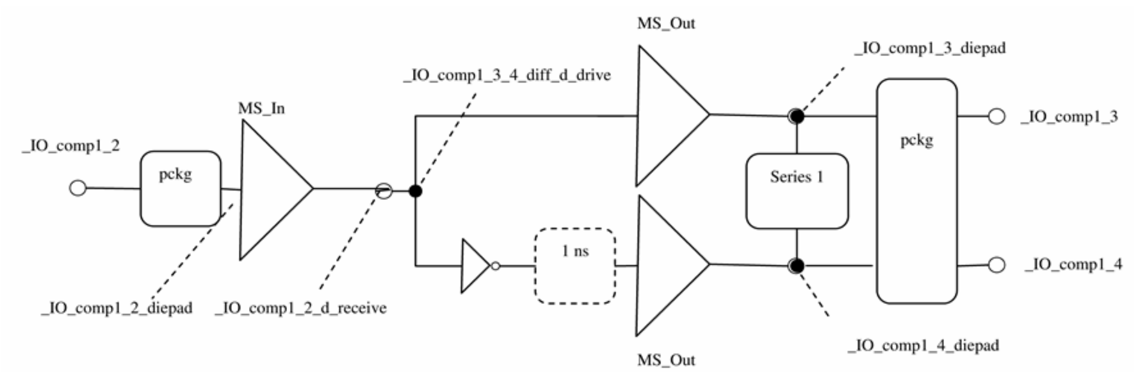


Figure 23-15 shows the Test_component of the *dummy.ibs* IBIS file in the “IBIS File Example” on page 1309. The component is instantiated with the name _IO_comp1. The digital output of the input buffer and the digital input of the output buffer are connected in the netlist by the user.

IBIS Package

Syntax

```
_IO_xx file="path" pkg_model="package_model_name"
```

Parameters

- **_IO_xx**

  IBIS instance name.

- **file=**"path"

  Specifies the path to the file that contains an IBIS formatted model.

  This can be the full path (for example, */user/test/Models/model.ibs*), a relative path, or just a file name. Both / and \ should be acceptable as separators in the path name (for the UNIX and NT versions). An Eldo option IBIS_SEARCH_PATH is available to specify the path to the directory to search for the IBIS files.

- **pkg_model=**"package_model_name"

  Specifies an IBIS package model name within the specified IBIS file. package_model_name is case-sensitive.

Equivalent subcircuit

For each IBIS instance that represents an IBIS package model, Eldo will create an equivalent subcircuit that contains sets of loss-less/lossy transmission lines and/or discrete RLC components.

Naming rules

External nodes of the equivalent subcircuit will be generated automatically by Eldo. Each pin in the package model will be represented by two external nodes; one at the die-pad side and the other one at the package-pin side. The naming of these pins follows the rules below:

- Node at die side:

  ```
  <IBIS_instance_name>_<pin_name>_DIEPAD
  ```

- Node at package side:

  ```
  <IBIS_instance_name>_<pin_name>
  ```

Examples

```
Package File Example

[IBIS Ver]  4.0
[File Name]  package_model_4pins.pkg
[File Rev]  1.0
[Define Package Model]  pm1
[Manufacturer]  ST None
```

```
[OEM]  Unknown
[Description]  None
[Number Of pins]  4
[pin Numbers]
pin1
pin2
pin3
pin4
[Model Data]
[Inductance Matrix]  Sparse_matrix
|
[Row]  pin1
pin1     418n
pin2     125n
|
[Row]  pin2
pin2     418n
pin3     125n
|
[Row]  pin3
pin3     418n
|
[Row]  pin4
pin4     418n
|
[Capacitance Matrix]  Sparse_matrix
|
[Row]  pin1
pin1     94p
pin2     -22p
|
[Row]  pin2
pin2     94p
pin3     -22p
|
[Row]  pin3
pin3     94p
|
[Row]  pin4
pin4     94p
|
[Resistance Matrix]  Sparse_matrix
|
[Row]  pin1
pin1     15
|
[Row]  pin2
pin2     15
|
[Row]  pin3
pin3     15
|
[Row]  pin4
pin4     15

|
[End Model Data]
[End Package Model]
```

```
[End]
```

The preceding package file example is referenced in the following netlist example.

- Netlist

```
_IO_pkg1
+ file=" package_model_4pins.pkg "
+ pkg_model=" pm1"
```

The above netlist instantiates the IBIS package located in the package_model_4pins.pkg IBIS file. The package model name is pm1. It is a four-pin package as shown in Figure 23-16. This package is described, in the package file, using the RLC matrix representation that accounts for the coupling between the pins. You can access the nodes that reside either inside or outside the package as shown.

Figure 23-16. Package model example

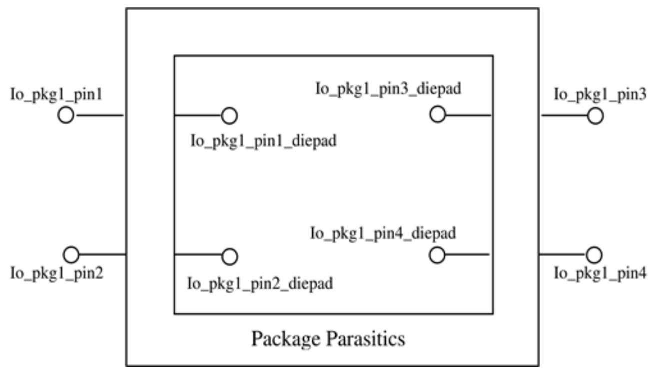


Figure 23-16 shows the package of the package_model_4pins.pkg IBIS file given in the "Package File Example" on page 1319. The package is instantiated with the name _IO_pkg1.

Electrical Board Description

Syntax

```
_IO_xx file="path" ebd_model="ebd_model_name"
+ [ref_des_map='ref_des_1=instance_1 [,…, ref_des_n=instance_n]']
```

Parameters

- **_IO_xx**

  IBIS instance name.

- **file**="path"

  Specifies the path to the file that contains an IBIS formatted model.

This can be the full path (for example, */user/test/Models/model.ibs*), a relative path, or just a file name. Both / and \ should be acceptable as separators in the path name (for the UNIX and NT versions). An Eldo option IBIS_SEARCH_PATH is available to specify the path to the directory to search for the IBIS files.

- **ebd_model=**"ebd_model_name"

  Specifies an IBIS Electrical Board Description (EBD) model name within the specified IBIS file. ebd_model_name is case-sensitive.

- **ref_des_map=**'ref_des_1=instance_1 [,…, ref_des_n=instance_n]'

  Maps an IBIS buffer component to a reference designator, where:

  - instance_n is the name of the Eldo IBIS instance used to describe an IBIS component.
  - ref_des_n is the reference designator visible in the Node sub-parameter of the [Path Description] keyword in the EBD file.

Note

Reference designator name and instance name are case insensitive.

ref_des_map can be repeated more than once in the _IO_ card.

If a reference designator is repeated in the ref_des_map, then the instance assigned to that reference designator will be that of the last one.

Equivalent subcircuit

For each IBIS instance that represents an IBIS EBD model, Eldo will create an equivalent subcircuit that contains sets of loss-less/lossy transmission lines and/or discrete RLC components.

Naming rules

External nodes of the equivalent subcircuit will be generated automatically by Eldo. Each pin in the EBD model, and each component pin, will be connected to the node in the netlist. The naming of these pins follows the rules below:

- Board pins:

  ```
  <IBIS_instance_name>_<pin_name>
  ```

- Component nodes:

  ```
  <mapped_ref_des>_<node_name>
  ```

Note

If a reference designator name is used within the IBIS file, but not mapped in the IBIS instance then the name used in the EBD file will be used without mapping.

Example

```
Example EBD File

[Ibis Ver]  3.1
[File Name] dummy.ebd
[File Rev]  0
[Source]    None
|
[Begin Board Description] EBD1
[Manufacturer] None
[Number Of Pins]  5
[Pin List] signal_name
1      S1
2      NC
3      POWER
4      GND
5      S2

[Path Description] 1
Pin 1
 Len = 0.66 L = 1.33333e-08 C = 2.08333e-12 /
 Len = 2.000 L = 1e-08 C = 1.77778e-12 /
Node Comp1.1

[Path Description] 5
Pin 5
 Len = 0.66 L = 1.33333e-08 C = 2.08333e-12 /
 Len = 2.000 L = 1e-08 C = 1.77778e-12 /
Node Comp2.1

[Reference Designator Map]
Comp1  dummy1.ibs component1
Comp2  dummy2.ibs component2

[End Board Description]
[End]
```

The preceding EBD file example is referenced in the following netlist example.

- Netlist

```
_IO_cmpnt1
+ file="dummy1.ibs"
+ component="component1"

_IO_cmpnt2
+ file="dummy2.ibs"
+ component="component2"

_IO_ebd1 file="dummy.ebd"
+ ebd_model="EBD1"
+ ref_des_map='u1=_IO_cmpnt1'
+ ref_des_map='u2=_IO_cmpnt2'
```

The above netlist instantiates two components and one EBD. The first component is located in the IBIS file *dummy1.ibs* and the second one in *dummy2.ibs*. These files are not shown here. The EBD is located in an EBD file *dummy.ebd*. The EBD model used is EBD1 given within the [Begin Board Description] and [End Board Description] keywords. The ref_des_map statements inform Eldo that _IO_cmpnt1 in the netlist corresponds to Comp1 in the EBD file, and that _IO_cmpnt2 corresponds to Comp2.

Supported Keywords and Sub-Parameters

Eldo can simulate IBIS devices either individually or in the overall component. Table 23-6 lists the supported keywords and sub-parameters in the current Eldo release. The IBISver column refers to the IBIS version in which the keyword/sub-parameter was introduced or modified.

Table 23-6. Supported Keywords and Sub-Parameters

Keyword	Sub-parameter	IBISver
[Package]	R_pkg, L_pkg & C_pkg	1.0
[Pin]		
	model_name	1.0
	R_pin, L_pin, C_pin	1.0
[Diff Pin]		
	inv_pin	2.1
	vdiff	3.1
	tdelay_typ	2.1
	tdelay_max	2.1
	tdelay_min	2.1
[Voltage Range]		1.0
[Pullup Reference]		2.1
[Pulldown Reference]		2.1
[POWER Clamp Reference]		2.1
[GND Clamp Reference]		2.1
[Pulldown]		1.0
[Pullup]		1.0
[POWER Clamp]		1.0
[GND Clamp]		1.0
[Ramp]		
	dV/dt_r	1.0
	dV/dt_f	1.0

Table 23-6. Supported Keywords and Sub-Parameters

Keyword	Sub-parameter	IBISver
	R_load	2.1
[Rising Waveform][1]	1000 Pts	4.0
[Falling Waveform][1]	1000 Pts	4.0
	C_fixture[2]	2.0
	R_fixture	2.0
	V_fixture	2.0
	V_fixture_min	2.1
	V_fixture_max	2.1
[Model]		
	Model_type	4.1
	Polarity	1.0
	Enable	1.0
	Vinl	2.0
	Vinh	2.0
	C_comp[1]	1.0
[Model Spec]		
	Vinh	3.2
	Vinl	3.2
	Vinh+	3.2
	Vinh-	3.2
	Vinl+	3.2
	Vinl-	3.2
[Rgnd]		2.1
[Rpower]		2.1
[Rac]		2.1
[Cac]		2.1
[R Series]		3.2
[L Series]		3.2
[C Series]		3.2
[Lc Series]		3.2

Table 23-6. Supported Keywords and Sub-Parameters

Keyword	Sub-parameter	IBISver
[Rc Series]		3.2
[Series Current]		3.2
[Series MOSFET]		3.2
	Vds	3.2
[On]		3.2
[Off]		3.2
[Series Switch Groups]	On, Off	3.2
[Series Pin Mapping]		
	pin_2	3.2
	model_name	3.2
	function_table_group	3.3
[Model Selector]		3.2
[Submodel]		
	Submodel_type	3.2
	Dynamic_clamp	3.2
	Bus_hold	3.2
[Submodel Spec]		3.2
	V_trigger_r	3.2
	V_trigger_f	3.2
	Off_delay	3.2
[GND Pulse Table]		3.2
[POWER Pulse Table]		3.2
	V_trigger_r	3.2
	V_trigger_f	3.2
[Pin Mapping]		
	pulldown_ref	2.1
	pullup_ref	2.1
	gnd_clamp_ref	2.1
	power_clamp_ref	2.1
[Driver Schedule]		

Table 23-6. Supported Keywords and Sub-Parameters

Keyword	Sub-parameter	IBISver
	Model_name	3.2
	Rise_on_dly	3.2
	Rise_off_dly	3.2
	Fall_on_dly	3.2
	Fall_off_dly	3.2
[External Model]	[End ...]	4.1
	Language	4.1
	SPICE	4.1
	Verilog-A	4.2
	Ports	4.1
	d_control	4.1
	d_drive	4.1
	d_enable	4.1
	d_receive	4.1
	a_puref	4.1
	a_pdref	4.1
	a_pcref	4.1
	a_gcref	4.1
	a_signal	4.1
	d_switch	4.1
	a_gnd	4.1
	a_pos	4.1
	a_neg	4.1
	a_signal_pos	4.1
	a_signal_neg	4.1
	D_to_A	4.1
	A_to_D	4.1
[External Circuit]	[End ...]	4.1
[Node Declarations]	[End ...]	4.1
[Circuit Call]	[End ...]	4.1

Table 23-6. Supported Keywords and Sub-Parameters

Keyword	Sub-parameter	IBISver
	Signal_pin	4.1
	Diff_signal_pins	4.1
	Series_pins	4.1
	Port_map	4.1
CIRCUITCALL	(reserved word)	4.1
[Package Model]		2.1
[Define Package Model]		2.1
[Manufacturer]		2.1
[OEM]		2.1
[Description]		2.1
[Number Of Sections]		3.0
[Number Of Pins]		2.1
[Pin Numbers]		2.1
	Len	3.0
	R	3.0
	L	3.0
	C	3.0
	Fork	3.0
	EndFork	3.0
[Model Data]	[End ...]	2.1
[Resistance Matrix]		2.1
[Inductance Matrix]		2.1
[Capacitance Matrix]		2.1
[Bandwidth]		2.1
[Row]		2.1
	Banded_matrix	2.1
	Sparse_matrix	2.1
	Full_matrix	2.1
[Begin Board Description]		3.0
[Manufacturer]		3.0

Table 23-6. Supported Keywords and Sub-Parameters

Keyword	Sub-parameter	IBISver
[Number Of Pins]		3.0
[Pin List]		3.0
	signal_name	3.0
[Path Description]		3.0
	Len	3.0
	R	3.0
	L	3.0
	C	3.0
	Fork	3.0
	EndFork	3.0
	Node	3.0
	Pin	3.0
[Reference Designator Map]		3.0

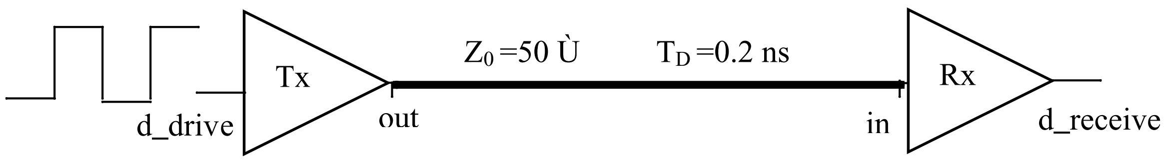
1. Eldo ignores waveforms that have L_fixture sub-parameter with non-zero values.
2. Eldo by default ignores waveforms that have C_fixture with non-zero values, to enable using these waveforms set IBIS instance keyword C_fixture=yes.

IBIS Tutorials

Tutorial 1—Single-Ended Tx-Rx System

This tutorial deals with a simple arrangement of a single-ended output buffer (Tx), transmission line, and a single-ended input buffer (Rx). This arrangement can be used for transmission line analysis mismatch as well as to assure that the Rx correctly recognizes the logic associated with the received signal levels.

Figure 23-17. Single-Ended Tx-Rx System



The transmission line has characteristics: impedance Z0=50 Ω and signal delay TD=0.2ns.

Complete Netlist

```
*Tutorial 1
_IO_Tx out d_drive
+file="tx_rx.ibs"
+component="TxRx"
+model="OUT"

TL out 0 in 0 Z0=50 TD=0.2ns

_IO_Rx in d_receive
+file="tx_rx.ibs"
+component="TxRx"
+model="IN"

Rterm d_receive 0 1e6

.param VHI=1.0
.param VLO =0.0
.param TDELAY=0
.param TRISE=10p
.param TFALL=Trise
.param TSAMPLE=5n
vstim d_drive 0 pattern VHI VLO TDELAY TRISE TFALL TSAMPLE 0 1 R

.tran 0 200n
.option step=10p
.plot tran v(out) v(d_drive) v(in) v(d_receive)
.end
```

Netlist Explanation

```
_IO_Tx out d_drive
+file="tx_rx.ibs"
+component="TxRx"
+model="OUT"
```

The above lines instantiate an IBIS output buffer Tx with output connected to out and digital input connected to d_drive. The output buffer is located in the *tx_rx.ibs* IBIS file within the TxRx component.

```
TL out 0 in 0 Z0=50 TD=0.2ns
```

Instantiates an ideal transmission line connected between the nodes out and in with ground reference. The transmission line has 50 Ω characteristics impedance and 0.2ns time delay.

```
_IO_Rx in d_receive
+file="tx_rx.ibs"
+component="TxRx"
+model="IN"
```

Instantiates an IBIS input buffer Rx with input connected to in and its digital output connected to d_receive. The input buffer is located in the *tx_rx.ibs* IBIS file within the TxRx component.

```
Rterm d_receive 0 1e6
```

Connects a 1 MΩ resistor between the digital output d_receive of the input buffer and ground. This has no impact on the results but avoids having a node with less than two connections.

```
.param VHI=1.0
.param VLO =0.0
.param TDELAY=0
.param TRISE=10p
.param TFALL=Trise
.param TSAMPLE=5n
vstim d_drive 0 pattern 1 0 0 10p 10p 5n 0 1 R
```

Instantiates the stimulus **v**stim for the system. The source is connected between the nodes d_drive and ground. It is defined using the **pattern** function with high voltage level of 1V, low voltage level of 0V, 0 starting delay time, 10 ps rise and fall times, 5ns sample time and finally it periodically alternates between 0 and 1.

The next part of the netlist specifies the simulation control directives indicating what type of simulation Eldo should perform on the circuit.

```
.tran 10p 200n
```

Specifies a transient analysis be performed lasting 200 ns with a plotting increment of 10 ps.

```
.option step=10p
```

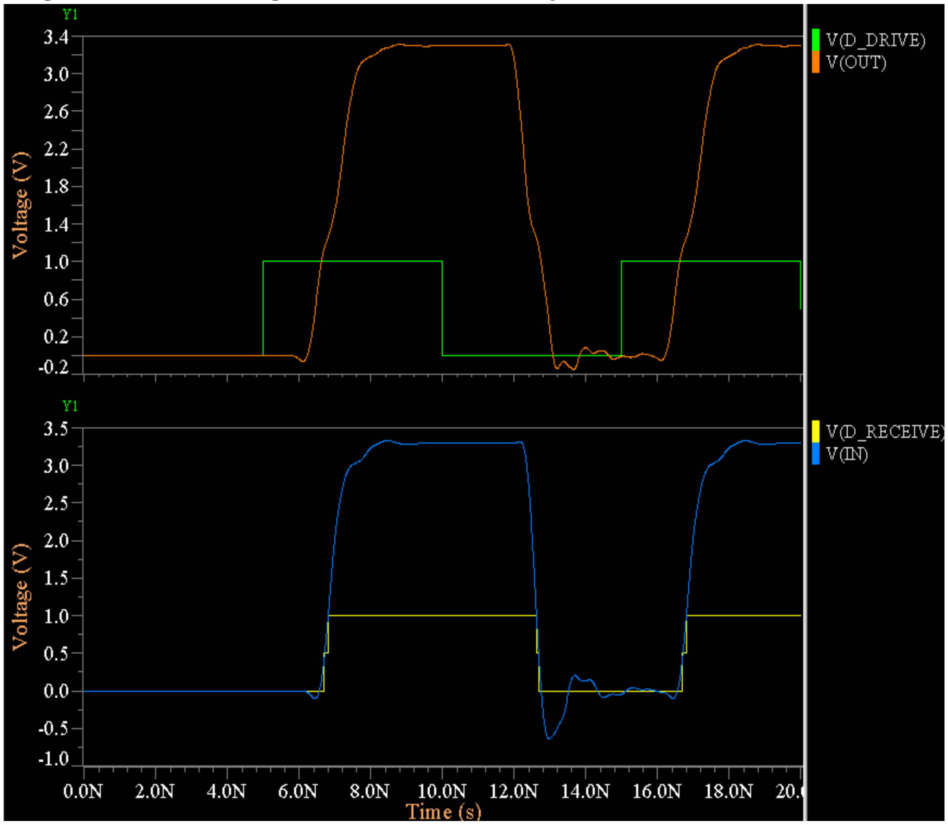
Imposes a time step of 10 ps.

```
.plot tran v(out) v(d_drive) v(in) v(d_receive)
```

Specifies voltage/time plots of the voltages at nodes out, d_drive, in and d_receive.

Simulation Results

Figure 23-18. Single-Ended Tx-Rx System Simulation Results



Ringing is clearly observed in the waveform at the receiver side. This is due to the mismatch between the TL Z0 and the Rx input impedance (ringing also means a mismatch exists between the Tx output impedance and the TL Z0). However, the Rx recognizes the signal logic levels and the digital output (d_receive) is given correctly without any undesired transitions due to the received signal ringing.

Tutorial 2—Pseudo-Differential Tx-Rx System

This tutorial deals with a simple arrangement of a pseudo-differential output buffer (Tx), transmission line and a pseudo-differential input buffer (Rx). Same as tutorial#1, this arrangement can be used for transmission line analysis mismatch as well as to assure that the differential Rx correctly recognizes the logic associated with the received differential signal levels.

Complete Netlist

```
*Tutorial 2
_IO_diffout outp outn d_drive
+ file="diffsource.ibs"
+ component="diffsource"
+ pin="1" inv_pin="2"

TLp outp 0 inp 0 Z0=50 TD=0.3ns
TLn outn 0 inn 0 Z0=50 TD=0.3ns

_IO_diffin inp inn d_receive
+ file="diffload.ibs"
+ component="diffload"
+ pin="1" inv_pin="2"

Rterm d_receive 0 1e6

.param VHI=1.0
.param VLO =0.0
.param TDELAYL=0
.param TRISE=300p
.param TFALL=Trise
.param TSAMPLE=3000p
vin D_drive 0 pattern VHI VLO TDELAYL TRISE TFALL TSAMPLE 1011010001 R

.tran 10p 40n

.plot tran v(d_drive) v(outp) v(outn) v(inp) V(inn) v(d_receive)

.end
```

Netlist Explanation

```
_IO_diffout outp outn d_drive
+ file="diffsource.ibs"
+ component="diffsource"
+ pin="1" inv_pin="2"
```

The above lines instantiate an IBIS differential output buffer diffout with non-inverting output connected to outp, inverting output connected to outn and digital input connected to d_drive. The output buffer is located in the *diffsource.ibs* IBIS file within the diffsource component. The differential buffer is recognized from the **pin** and **inv_pin** keywords. Pin names given using these keywords must match those in the *diffsource.ibs* IBIS file under the [Diff_pin] keyword.

```
TLp outp 0 inp 0 Z0=50 TD=0.3ns
TLn outn 0 inn 0 Z0=50 TD=0.3ns
```

Instantiates a pair of ideal transmission lines, one of them connected between the nodes outp and inp with ground reference, and the second one connected between the nodes outn and inn with ground reference. Both transmission lines have 50 Ω characteristics impedance and 0.2ns time delay.

```
_IO_diffin inp inn d_receive
+ file="diffload.ibs"
```

```
+ component="diffload"
+ pin="1" inv_pin="2"
```

Instantiates an IBIS differential input buffer diffin with non-inverting input connected to inp, inverting input connected to inn and digital output connected to d_receive. The input buffer is located in the *diffload.ibs* IBIS file within the diffload component. The differential buffer is recognized from the **pin** and **inv_pin** keywords. Pin names given using these keywords must match those in the *diffload.ibs* IBIS file under the [Diff_pin] keyword.

```
Rterm d_receive 0 1e6
```

Connects a 1 MΩ resistor between the digital output d_receive of the input buffer and ground. This has no impact on the results avoids having a node with less than two connections.

```
.param VHI=1.0
.param VLO =0.0
.param TDELAYL=0
.param TRISE=300p
.param TFALL=Trise
.param TSAMPLE=3000p
vin d_drive 0 pattern VHI VLO TDELAYL TRISE TFALL TSAMPLE 1011010001 R
```

Instantiates the stimulus **v**in for the system. The source is connected between the nodes d_drive and ground. It is defined using the **pattern** function with high voltage level of 1V, low voltage level of 0V, 0 starting delay time, 300 ps rise and fall times, 3ns sample time and have the pattern 1011010001.

The next part of the netlist specifies the simulation control directives indicating what type of simulation Eldo should perform on the circuit.

```
.tran 10p 40n
```

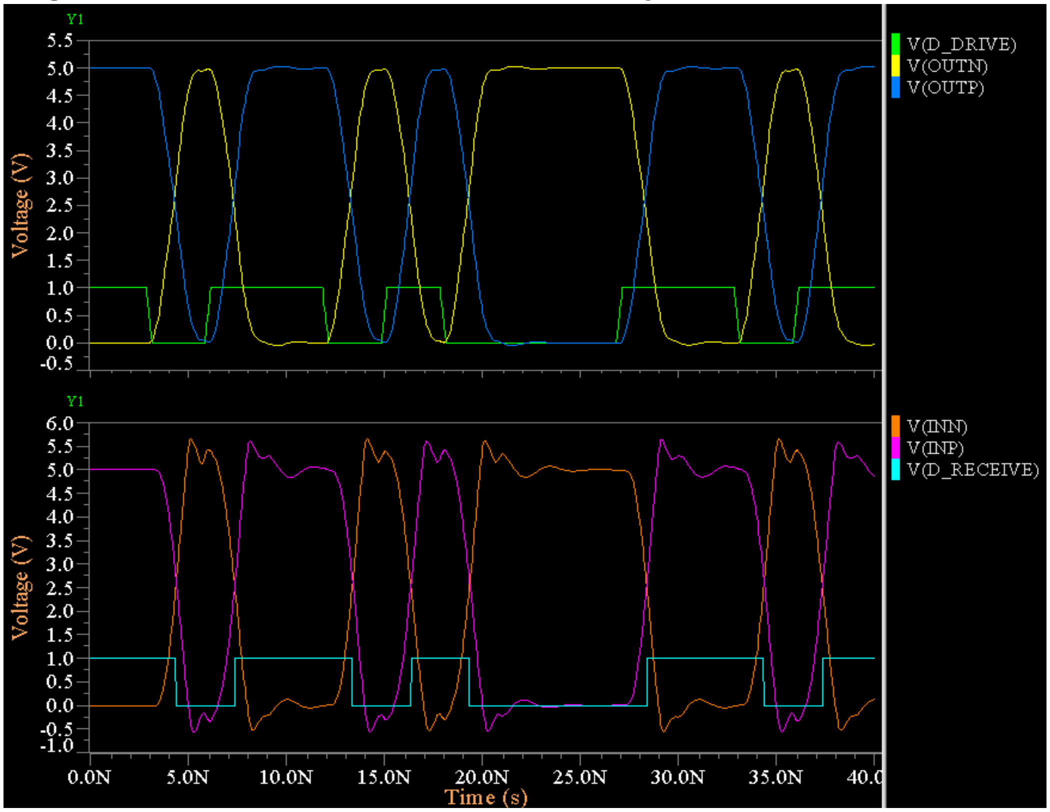
Specifies a transient analysis be performed lasting 40 ns with a plotting increment of 10 ps.

```
.plot tran v(d_drive) v(outp) v(outn) v(inp) V(inn) v(d_receive)
```

Specifies voltage/time plots of the voltages at nodes d_drive, outp, outn, inp, inn and d_receive.

Simulation Results

Figure 23-19. Pseudo-Differential Tx-Rx System Simulation Results



Tutorial 3—Package Parasitics Cross Talk

This tutorial deals with the simulation of an entire component and observation of the effect of cross talk between the pins of the package. This is one of the important issues in signal integrity analysis.

Complete Netlist

```
*Tutorial 3
_IO_Trx
+file="tx_rx.ibs"
+component="transceiver"
+package=3

vp _IO_TRX_5 0 pwl (0 0 1n 0 1.6n 3.3 8n 3.3 8.6n 0)


vcc _IO_TRX_4 0 3.3
.connect _IO_TRX_1_d_drive 0
.connect _IO_TRX_2_d_drive 0
.connect _IO_TRX_3_d_drive 0
.connect _IO_TRX_8 0
Rterm1 _IO_TRX_5_d_receive 0 1e6
Rterm2 _IO_TRX_6_d_receive 0 1e6
```

```
Rterm3 _IO_TRX_7_d_receive 0 1e6

.tran 1p 14n

.option step=1p

.plot tran v(_IO_TRX_5) v(_IO_TRX_6)

.end
```

Netlist Explanation

```
_IO_Trx
+file="tx_rx.ibs"
+component="transceiver"
+package=3
```

The above lines instantiate an IBIS component. Component instantiation means all the buffers inside the component as well as coupling between pins can be simulated. The component is named transceiver and located in the *tx_rx.ibs* IBIS file. The **package** parameter in the _IO element specifies the package modeling type to be used. In our case, package=3 means that the advanced package model, located in the *tx_rx.ibs* IBIS file under [Define package model] will be used.

```
vp _IO_Trx_5 0 pwl (0 0 1n 0 1.6n 3.3 8n 3.3 8.6n 0)
```

Instantiates the stimulus **v**p which is a pulse waveform connected between pin 5 of the IBIS component and ground. It is important to note that all other component pins exist in the simulation but left unconnected here. A warning will appear about these unconnected pins.

```
vcc _IO_TRX_4 0 3.3
.connect _IO_TRX_1_d_drive 0
.connect _IO_TRX_2_d_drive 0
.connect _IO_TRX_3_d_drive 0
.connect _IO_TRX_8 0
Rterm1 _IO_TRX_5_d_receive 0 1e6
Rterm2 _IO_TRX_6_d_receive 0 1e6
Rterm3 _IO_TRX_7_d_receive 0 1e6
```

The above lines have no impact on the simulation results. They are used to avoid warnings about nodes that have less than two connections. A voltage source **v**cc is connected between the node _IO_TRX_4 and ground. The digital inputs of the output buffers (_IO_TRX_1_d_drive, _IO_TRX_2_d_drive and _IO_TRX_3_d_drive) are tied to ground using the .connect statement. Also the node _IO_TRX_8 is connected to ground. Then 1 MΩ resistors are connected between the digital outputs of the input buffers (_IO_TRX_5_d_receive, _IO_TRX_6_d_receive and _IO_TRX_7_d_receive) and ground.

The next part of the netlist specifies the simulation control directives indicating what type of simulation Eldo should perform on the circuit.

```
.tran 1p 14n
```

Specifies a transient analysis be performed lasting 14 ns with a plotting increment of 1 ps.

```
.option step=1p
```

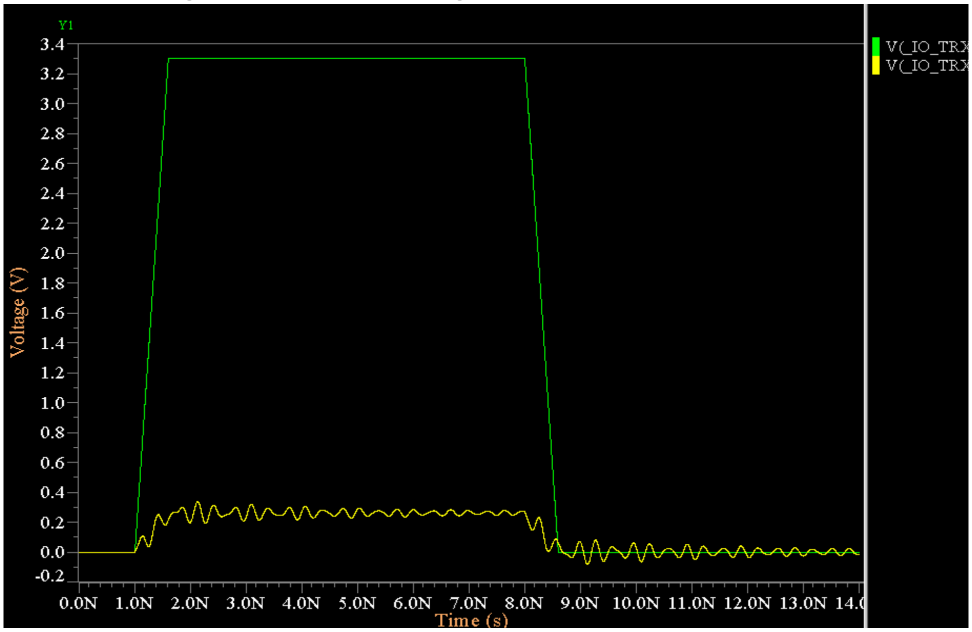
Imposes a time step of 1 ps. This small time step is imposed to capture the ringing that will occur on pin 6 due to both capacitive and inductive coupling with pin 5.

```
.plot tran v(_IO_TRX_5) v(_IO_TRX_6)
```

Specifies voltage/time plots of the voltages at both pin 5 and 6 of the IBIS component.

Simulation Results

Figure 23-20. Package Parasitics Cross Talk

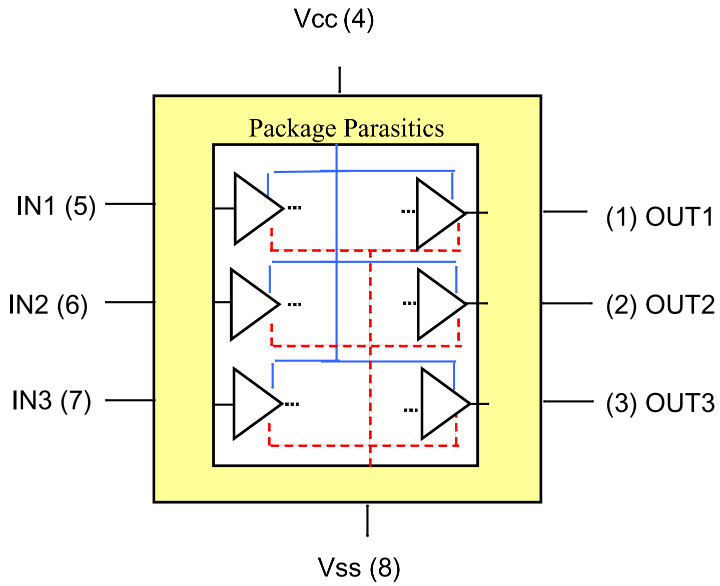


There is both capacitive and inductive coupling between pins 5 and 6. This coupling results in a ringing effect on pin 6.

Tutorial 4—Simultaneous Switching Output Noise

This tutorial deals with the simulation simultaneous switching output noise (SSON). The [Pin Mapping] keyword in the IBIS file contains information on how power supplies are connected to individual buffers or groups of buffers; it can be used for predicting SSON.

Figure 23-21. Component Internal Connection



Complete Netlist

```
*Tutorial4
_IO_comp
+file="tx_rx_pm.ibs"
+component="transceiver_pm"
+package=1

VCC _IO_comp_4 0 3.3
.connect _IO_comp_8 0

Vin1 _IO_comp_5 0 0
Vin2 _IO_comp_6 0 3.3
.param VHI=3.3
.param VLO =0.0
.param TDELAYL=1n
.param TRISE=0.1n
.param TFALL=Trise
.param TSAMPLE=5n
Vin3 _IO_comp_7 0 pattern VHI VLO TDELAYL TRISE TFALL TSAMPLE 0 1  R

.connect _IO_comp_5_d_receive _IO_comp_1_d_drive
.connect _IO_comp_6_d_receive _IO_comp_2_d_drive
.connect _IO_comp_7_d_receive _IO_comp_3_d_drive

.tran 10p 50n
.option step=10p
.plot tran v(_IO_comp_1) v(_IO_comp_2) v(_IO_comp_3)
.end
```

Netlist Explanation

```
_IO_comp
+file="tx_rx_pm.ibs"
+component="transceiver_pm"
+package=1
```

The above lines instantiate an IBIS component. The component is named transceiver_pm and located in the *tx_rx_pm.ibs* IBIS file. Package=1 means that package parasitics provided in the IBIS file under [Package] keyword, where package pins are assumed uncoupled, will be used.

```
vcc _IO_comp_4 0 3.3
.connect _IO_comp_8 0
```

Connects the power supply of the component. Note the user should connect the power supply manually, if the competent has a [Pin Mapping] keyword. A DC voltage source of 3.3 V is connected to pin 4 of the component and pin 8 is connected to ground.

```
vin1 _IO_comp_5 0 0
vin2 _IO_comp_6 0 3.3
.param VHI=3.3
.param VLO =0.0
.param TDELAYL=1n
.param TRISE=0.1n
.param TFALL=Trise
.param TSAMPLE=5n
vin3 _IO_comp_7 0 pattern VHI VLO TDELAYL TRISE TFALL TSAMPLE 0 1 1 0  R
```

Defines the inputs of the component input buffers. Pin 5 is connected to 0 voltage, pin 6 is connected to a DC voltage of 3.3 V and pin 7 is connected to the stimulus of the system **v**in3. It is defined using the **pattern** function with high voltage level of 3.3V, low voltage level of 0V, 1 ns starting delay time, 0.1 ns rise and fall times, 5 ns sample time and have the pattern 0110.

```
.connect _IO_comp_5_d_receive _IO_comp_1_d_drive
.connect _IO_comp_6_d_receive _IO_comp_2_d_drive
.connect _IO_comp_7_d_receive _IO_comp_3_d_drive
```

Connects the input buffers digital output to the output buffers digital input.

The next part of the netlist specifies the simulation control directives indicating what type of simulation Eldo should perform on the circuit.

```
.tran 10p 50n
```

Specifies a transient analysis be performed lasting 50 ns with a plotting increment of 10 ps.

```
.option step=10p
```

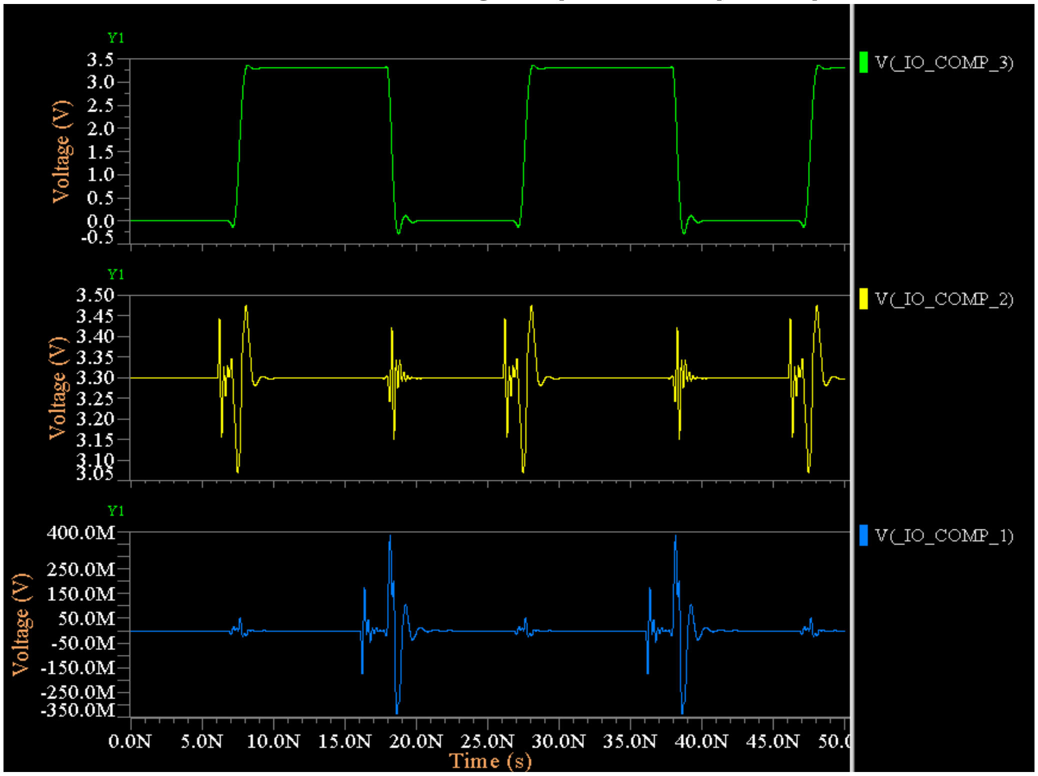
Imposes a time step of 10 ps.

```
.plot tran v(_IO_comp_1) v(_IO_comp_2) v(_IO_comp_3)
```

Specifies voltage/time plots of the voltages at both pin 1, 2, and 3 of the IBIS component.

Simulation Results

Figure 23-22. Simultaneous Switching Output Noise (SSON) Simulation Results



A difference is observed in the results from the case when package=1 was set (compare yellow and blue waveforms). The time varying signal at OUT1 and OUT2 is not only due to IR and LdI/dt voltage drops on the supply pin, but in addition has the capacitive and inductive coupling between the package pins.

Chapter 24
Tutorials

Introduction

The most productive way of learning a simulation tool such as Eldo is to get 'hands-on' experience by sitting at a terminal and working through, stage by stage, a number of practical circuit simulation examples and tutorials. This chapter has been written to achieve this.

The tutorials cover a wide range of simple but concise circuit applications. Thus, they should be of interest not only to the novice user, but also to the experienced user wishing to learn more about specific simulation techniques within Eldo.

Upon completion, a wide range of techniques needed to perform efficient and productive analysis using Eldo should have been learnt.

Each tutorial starts with a brief description of the circuit in question together with a circuit diagram. A short summary of the Eldo commands used within the tutorial follows this, in order to aid users wishing to learn about a specific topic or the use of certain commands within Eldo. A complete circuit netlist is then shown, followed by a breakdown and explanation of each section. Actual output results from the circuit conclude each tutorial using EZwave, the Eldo waveform viewer.

A summary of the circuits used in this chapter, together with a brief description of the Eldo subject areas dealt with are listed below. Listings for these examples may be found in the following subdirectories included with your software:

```
$MGC_AMS_HOME/examples/eldo
```

where `$MGC_AMS_HOME` is the directory where the software resides.

Note

For more examples please refer to the Examples appendix and "Examples for IEM" on page 1102.

Table 24-1. Tutorials

Tutorial	Circuit Name	Eldo Description
Tutorial #1—Parallel LCR Circuit	*parallel_lcr.cir*	General introduction AC analysis

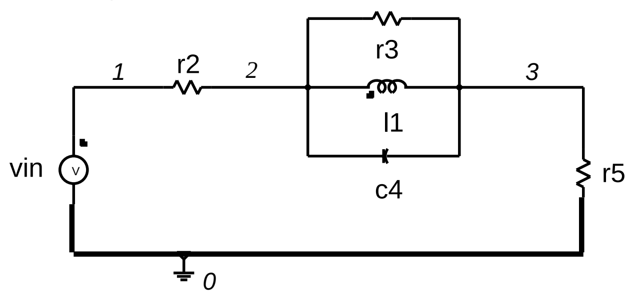
Table 24-1. Tutorials

Tutorial	Circuit Name	Eldo Description
Tutorial #2—4th Order Butterworth Filter	*butterworth.cir*	Transient & AC analysis
Tutorial #3—Band Pass Filter	*bandpass.cir*	AC & Noise analysis Model description
Tutorial #4—Low Pass Filter	*lowpass.cir*	AC analysis. Subcircuit definition
Tutorial #5—Colpitts Oscillator	*colpitts.cir*	Transient analysis. Use of model library files
Tutorial #6—High Voltage Cascade	*hv_cascade.cir*	Transient analysis. Model description
Tutorial #7—Non-inverting Amplifier	*noninvert_amp.cir*	AC & Monte Carlo analysis. Model description
Tutorial #8—Bipolar Amplifier	*bip_amplifier.cir*	DC sensitivity analysis
Tutorial #9—SC Low Pass Filter	*sc_lowpass.cir*	Transient and small signal AC analyses using the Z-domain switched capacitor models

Tutorial #1—Parallel LCR Circuit

This simple circuit simulation gives a general introduction to the syntax of Eldo by performing an AC analysis on a parallel LCR circuit. The complete netlist can be found in the file *parallel_lcr.cir*.

Figure 24-1. Parallel LCR Circuit



Summary of Eldo Commands used in this Tutorial

.AC—AC analysis
.OPTION—Simulator configuration
.PLOT—Plot simulator results

Complete Netlist

```
LCR Parallel Network
vin 1 0 ac 10
r2 1 2 50
r3 2 3 50k
r5 3 0 50
l1 2 3 100u
c4 2 3 10n
.ac dec 10 1 1g
.option eps = 1.0e-6
.plot ac vdb(2) vdb(3) (-30,20)
.end
```

Netlist Explanation

```
LCR Parallel Network
```

The above line is the circuit title. It must always be the first line of a simulation.

The first part of the Eldo netlist is a description of the circuit components.

```
vin 1 0 ac 10
```

The above line defines an AC voltage source `vin` connected between the nodes `1` and `0` of value 10V.

```
r2 1 2 50
r3 2 3 50k
r5 3 0 50
l1 2 3 100u
c4 2 3 10n
```

The above lines define the devices present in the circuit to be simulated. Each device instantiation gives the component name, the nodes to which the component is connected and the value of the component.

The next part of the netlist specifies the simulation control directives indicating what type of simulation Eldo should perform on the circuit.

```
.ac dec 10 1 1g
```

The above line indicates that an AC analysis should be performed on the circuit within the frequency range 1Hz to 1 GHz with 10 steps per decade.

```
.option eps = 1.0e-6
```

The above line increases the internal accuracy of Eldo from its default value of 5mV to 1µV. This is very important to achieve the best results for the simulation of most analog circuits.

```
.plot ac vdb(2) vdb(3) (-30,20)
```

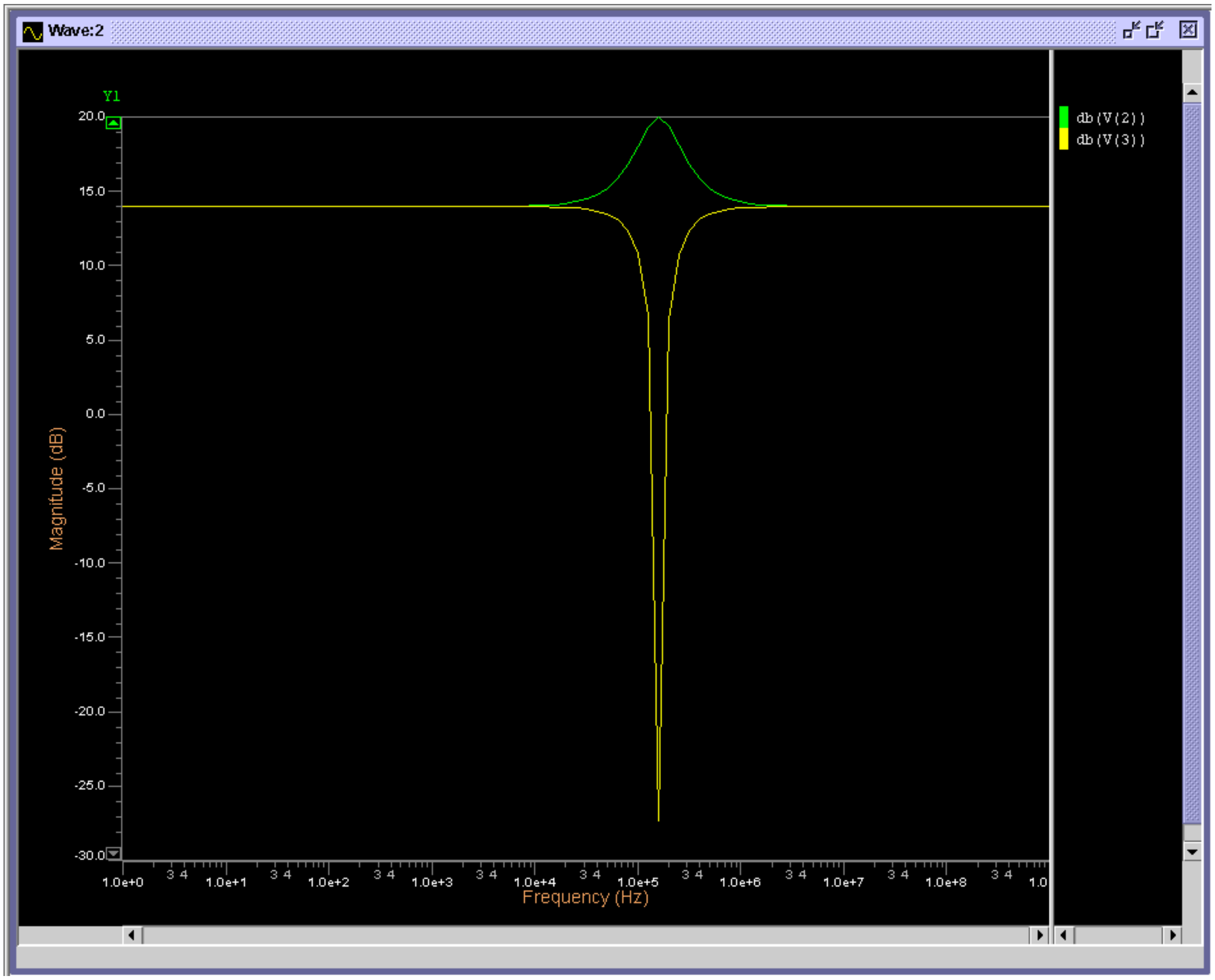
The above line specifies a dB/frequency plot of the nodes `2` and `3` on the same graph between the limits –30 dB and +20 dB. The results are stored in the *parallel_lcr.wdb* file and can be displayed using the EZwave graphical results post-processor.

```
.end
```

The netlist must always be terminated with the above command.

Simulation Results

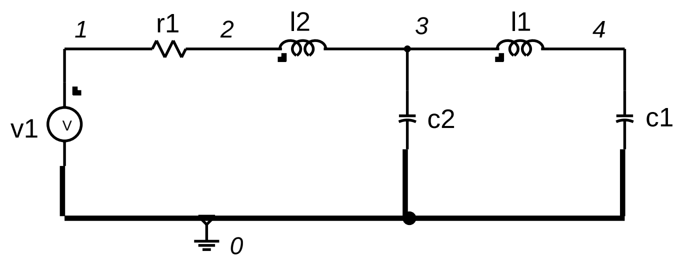
Figure 24-2. Tutorial #1—Simulation Results



Tutorial #2—4th Order Butterworth Filter

This example simulates a 4th order Butterworth filter with a transient and an AC analysis being performed on the circuit. The complete netlist can be found in the file *butterworth.cir*.

Figure 24-3. 4th Order Butterworth Filter



Summary of Eldo Commands used in this Tutorial

.AC—AC analysis
.PLOT—Plot simulator results
.TRAN—Transient analysis

Complete Netlist

```
4th Order Butterworth Filter
c1 4 0 1.5307n
c2 3 0 1.0824n
l1 3 4 1.5772u
l2 2 3 .38268u
r1 1 2 1
v1 1 0 ac 1 pwl (0 0 1u 0 2u 1 20u 1 20.1u 0)
.tran .2u 40u
.plot tran v(4) (-1,1.5)
.plot tran v(1) (0,1.5)
.ac dec 20 10000 100meg
.option eps=1.0e-6 be
.plot ac vdb(4) (-120,40)
.plot ac vp(4) (-200,200)
.end
```

Netlist Explanation

```
4th Order Butterworth Filter
c1 4 0 1.5307n
c2 3 0 1.0824n
l1 3 4 1.5772u
l2 2 3 .38268u
r1 1 2 1
v1 1 0 ac 1 pwl (0 0 1u 0 2u 1 20u 1 20.1u 0)
```

The above line defines an AC source of 1V and a time dependent Piece Wise Linear function between the nodes `1` and `0`. The **`pwl`** parameters describe a signal that stays at 0V until 1µs where it rises to 1V in 1µs. The signal stays at 1V until 20µs where it drops back to 0V in 0.1µs.

Refer to the output results for a pictorial representation of this signal.

```
.tran .2u 40u
```

The above line specifies that a transient analysis should be performed on the circuit lasting 40µs with a plotting increment for the line printer of 0.2µs.

```
.plot tran v(4) (-1,1.5)
.plot tran v(1) (0,1.5)
```

The above lines specify that voltage/time plots should be performed on separate graphs of the voltage at node `4` between the limits –1 and +1.5V, and of the voltage at node `1` between the limits 0 and +1.5V.

```
.ac dec 20 10000 100meg
```

The above line indicates that an AC analysis should be performed on the circuit within the frequency range 10000Hz to 100MHz with 20 steps per decade. This AC analysis statement replaces the transient analysis definition found earlier in the netlist.

```
.option eps=1.0e-6 be
```

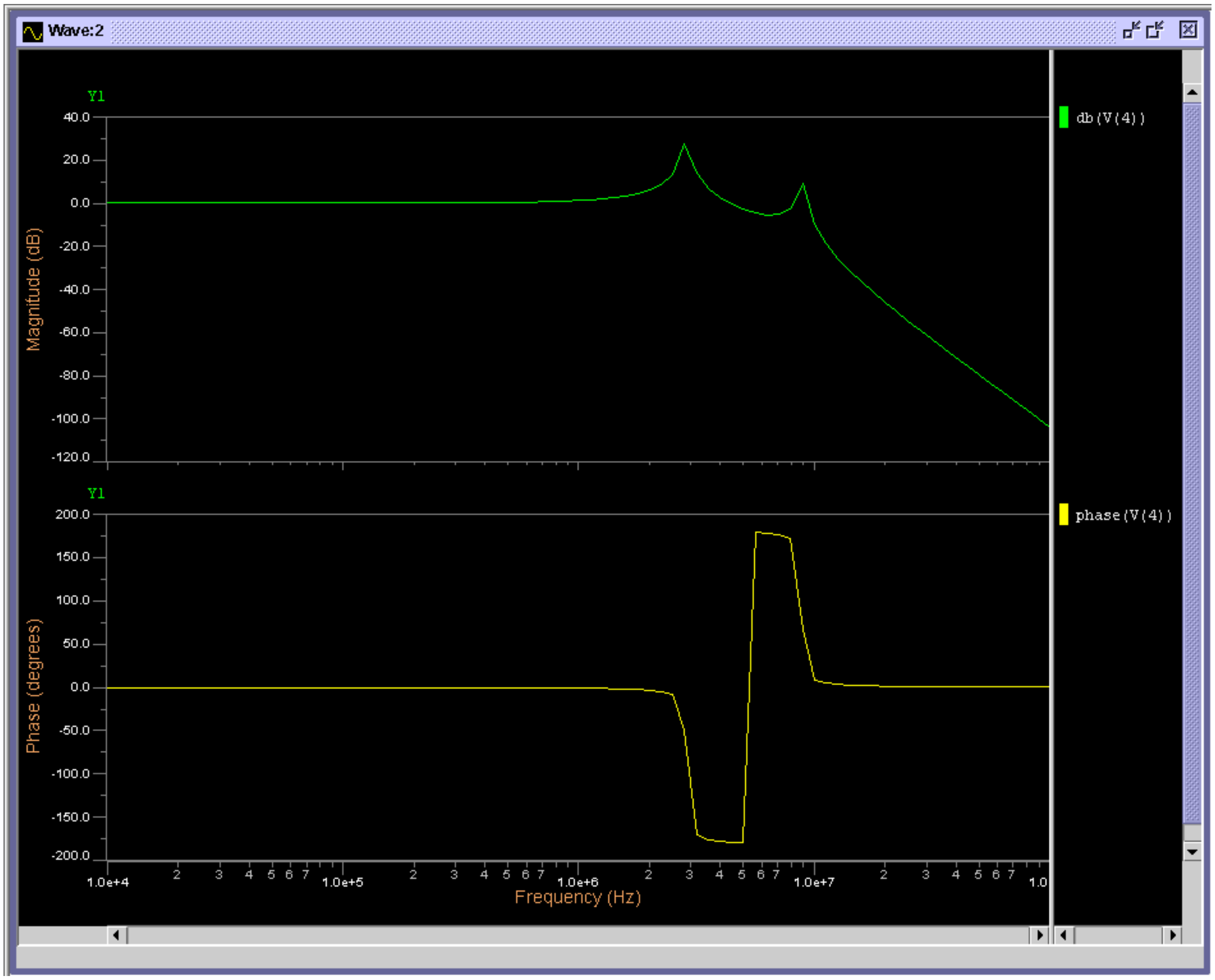
The above line sets the simulator accuracy together with the simulator algorithm as Backward Euler.

```
.plot ac vdb(4) (-120,40)
.plot ac vp(4) (-200,200)
```

The above lines specify that dB/frequency and phase/frequency plots should be performed of the voltage at node `4` between the limits –120dB and +40dB and -200 and +200 degrees respectively. These commands are added to the first simulation run netlist. The results are also added to the file *butterworth.wdb* and can be displayed as a second simulation page using the EZwave graphical post-processor.

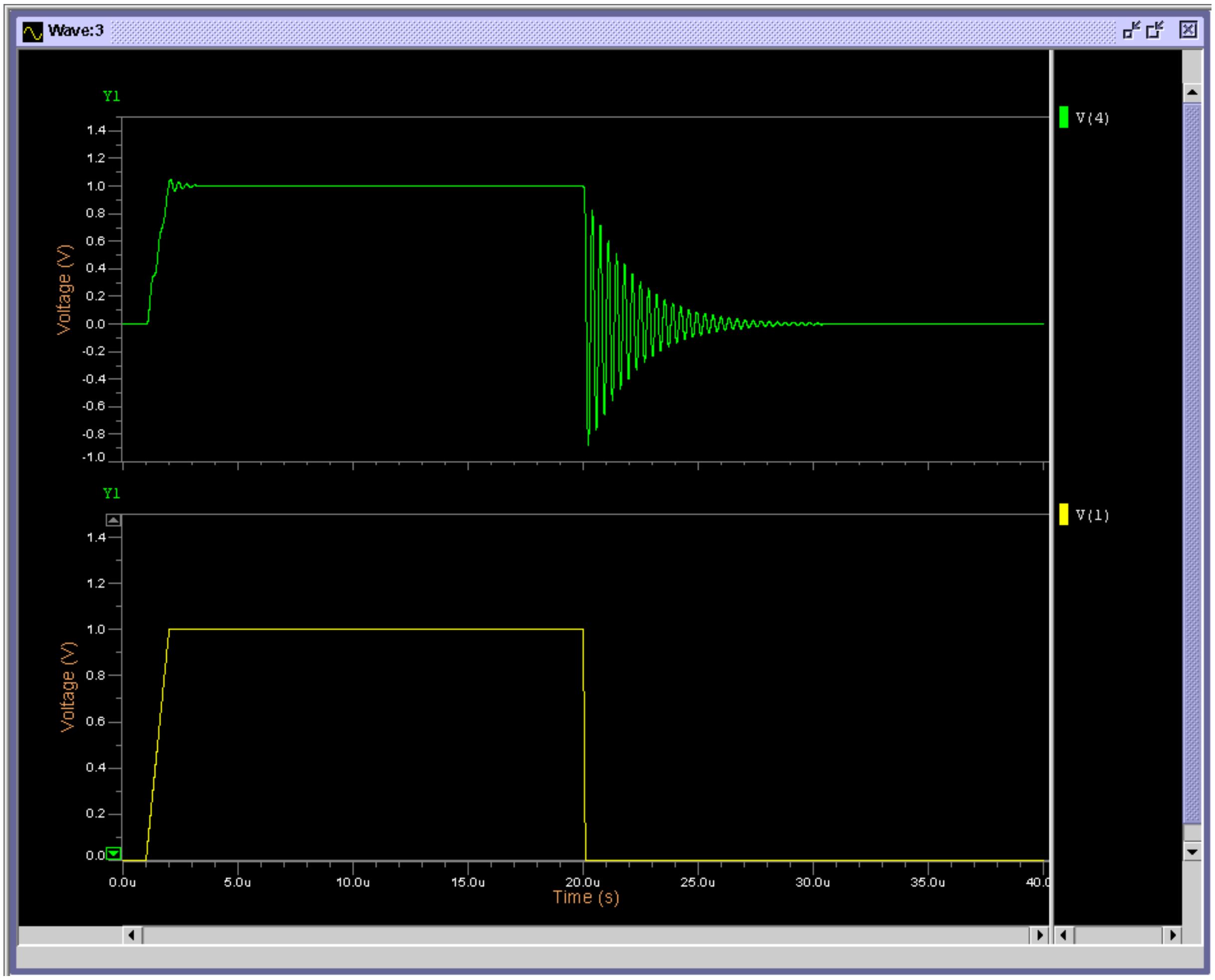
Simulation Results—1

Figure 24-4. Tutorial #2—Simulation Results—1



Simulation Results—2

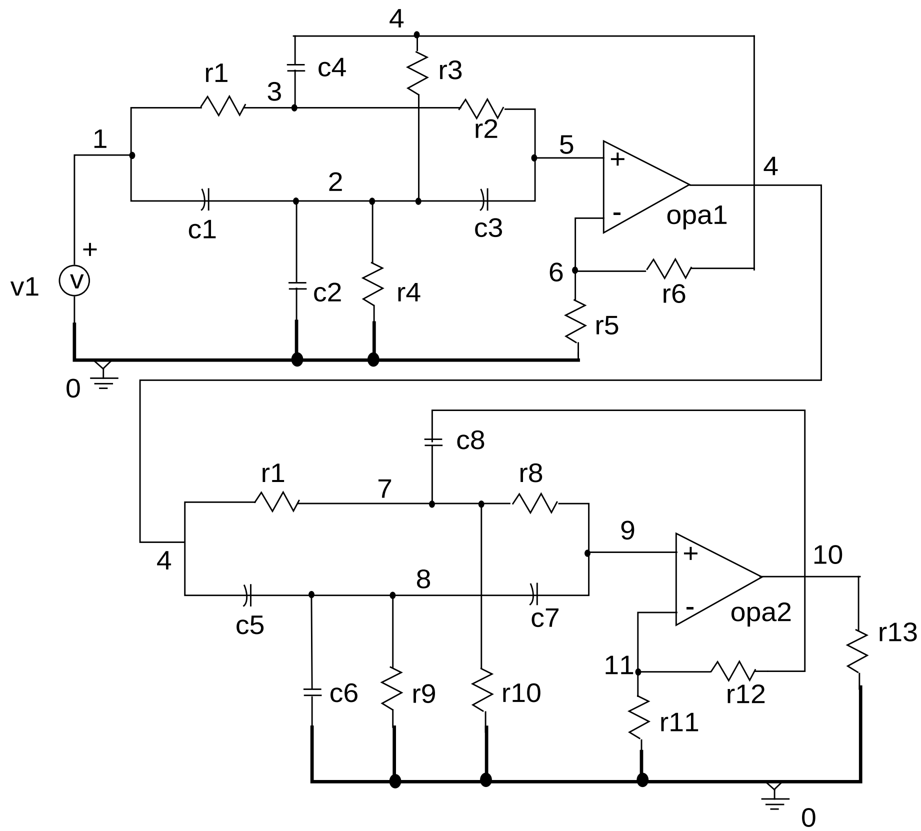
Figure 24-5. Tutorial #2—Simulation Results—2



Tutorial #3—Band Pass Filter

This tutorial deals with an op-amp band pass filter. The simulation performs an AC analysis of the circuit, together with a noise analysis of the output stage of the filter. The complete netlist can be found in the file *bandpass.cir*.

Figure 24-6. Band Pass Filter



Summary of Eldo Commands used in this Tutorial

.MODEL—Model definition
.NOISE—Noise analysis

Complete Netlist

```
Band-Pass filter
.model ampop modfas gain=10000.0 p1=5e3
r1 1 3 10k
r2 3 5 10k
r3 2 4 13.95k
r4 2 0 7.79k
r5 6 0 10k
r6 4 6 3.9k
r7 4 7 244.7k
r8 7 9 10k
r9 8 0 5k
r10 7 0 10.43k
r11 11 0 10k
r12 11 10 8.87k
r13 10 0 50
c1 1 2 3.27n
c2 2 0 16.73n
c3 2 5 20n
c4 3 4 40n
c5 4 8 3.17n
c6 8 0 9.5n
c7 8 9 12.7n
c8 7 10 25.3n
y1 opamp2 pin: 5 6 4 0 model: ampop
y2 opamp2 pin: 9 11 10 0 model: ampop
v1 1 0 ac
.ac dec 80 100 10k
.noise v(10) v1 80
.plot noise db(inoise)
.plot noise db(onoise)
.plot ac vdb(10) (10,-50)
.end
```

Netlist Explanation

```
Band-Pass filter
.model ampop modfas gain=10000.0 p1=5e3
```

The above line describes the electrical parameters of the user defined model `ampop` based on the `opamp2` macromodel. The gain (**`gain`**) and dominant pole frequency (**`p1`**) of the model are set to 10000 and 5×10^3Hz.

For more details on the `opamp2` macromodel and its parameters, refer to Analog Macromodels.

```
v1 1 0 ac
r1 1 3 10k
r2 3 5 10k
r3 2 4 13.95k
r4 2 0 7.79k
r5 6 0 10k
r6 4 6 3.9k
r7 4 7 244.7k
r8 7 9 10k
r9 8 0 5k
r10 7 0 10.43k
r11 11 0 10k
r12 11 10 8.87k
r13 10 0 50
c1 1 2 3.27n
c2 2 0 16.73n
c3 2 5 20n
c4 3 4 40n
c5 4 8 3.17n
c6 8 0 9.5n
c7 8 9 12.7n
c8 7 10 25.3n
yopa1 opamp2 pin: 5 6 4 0 model: ampop
yopa2 opamp2 pin: 9 11 10 0 model: ampop
```

The above lines instantiate two operational amplifiers `yopa1` and `yopa2` of macromodel type `opamp2` (linear 2-pole) connected between the nodes `5`, `6`, `4` and `0` and between the nodes `9`, `11`, `10` and `0` respectively. The electrical parameters of the macromodel are defined in the model `ampop`.

```
.ac dec 80 100 10k
```

The above line indicates that an AC analysis should be performed on the circuit within the frequency range 100 Hz to 10 kHz with 80 steps per decade.

```
.noise v(10) v1 80
```

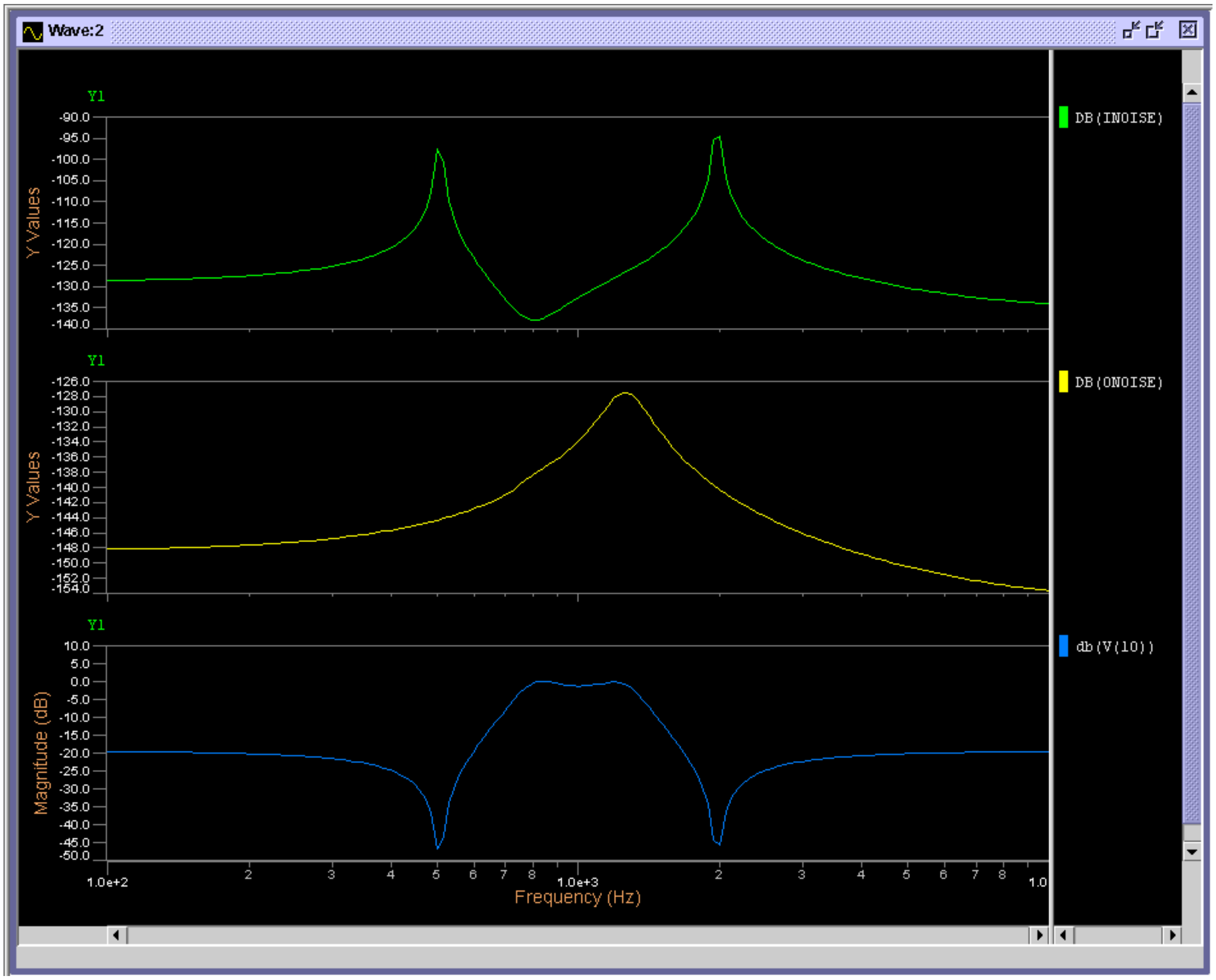
The above line indicates that a noise analysis should be performed of the voltage at node 10 with the voltage source `v1` as input noise voltage. The analysis should be averaged over 80 frequency points.

```
.plot noise db(inoise)
.plot noise db(onoise)
.plot ac vdb(10) (10,-50)
```

The above lines specify a dB/frequency plot to be performed on the input and output noise, together with a dB/frequency plot of the voltage at node 10, the output stage of the filter, between the limits 10 and –50dB.

Simulation Results

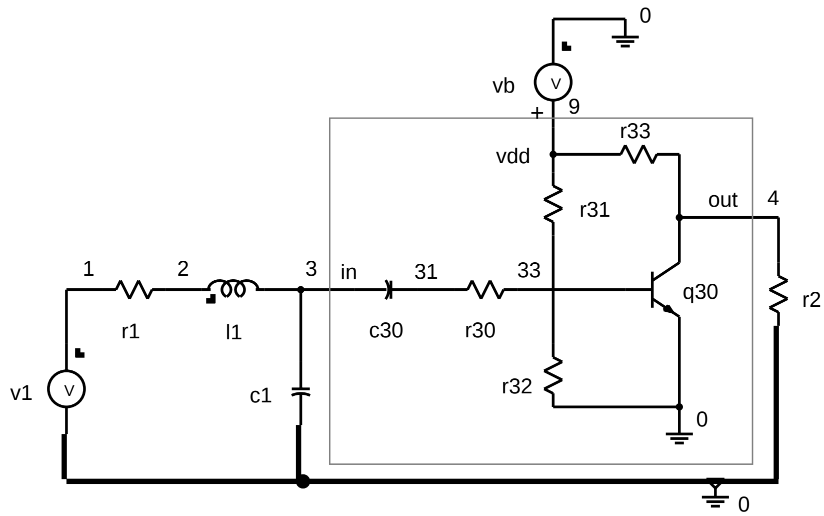
Figure 24-7. Tutorial #3—Simulation Results



Tutorial #4—Low Pass Filter

This tutorial deals with the AC analysis of a low pass filter which incorporates a voltage amplifier, illustrating the use of Eldo's subcircuit capabilities, as the voltage amplifier part of the circuit itself is defined in this manner. The complete netlist can be found in the file *lowpass.cir*.

Figure 24-8. Low Pass Filter



Summary of Eldo Commands used in this Tutorial

.AC—AC analysis
.MODEL—Model definition
.PLOT—Plot simulator results
.SUBCKT—Subcircuit definition

Complete Netlist

```
Low Pass Filter incorporating a Voltage Amplifier
* .MODEL definition
.model q2n2222 npn is=1.9e-14 bf=150 vaf=100 ikf=.175
+ ise=5e-11 ne=2.5 br=7.5 var=6.38 ikr=.012 isc=1.9e-13
+ nc=1.2 rc=.4 cje=26p tf=.5e-9 cjc=11p tr=30e-9 xtb=1.5
+ kf=3.2e-16 af=1.0
* Subcircuit definition
.subckt amp in out vdd
c30 in 31 47u
r30 31 33 390
r31 vdd 33 50k
r32 33 0 15k
q30 out 33 0 q2n2222
r33 vdd out 750
.ends amp
```

```
r1 1 2 10
l1 2 3 1.3m
c1 3 0 100n
r2 4 0 50k
x1 3 4 9 amp
vb 9 0 5
v1 1 0 ac 1
* Commands
.ac dec 10 1 1g
.plot ac vdb(3) vdb(4) (-250,50)
.plot ac vp(3) vp(4) (-200,200)
.end
```

Netlist Explanation

```
Low Pass Filter incorporating a Voltage Amplifier
.model q2n2222 npn is=1.9e-14 bf=150 vaf=100 ikf=.175
+ ise=5e-11 ne=2.5 br=7.5 var=6.38 ikr=.012 isc=1.9e-13
+ nc=1.2 rc=.4 cje=26p tf=.5e-9 cjc=11p tr=30e-9 xtb=1.5
+ kf=3.2e-16 af=1.0
.subckt amp in out vdd
```

The above line indicates the start of the voltage amplifier subcircuit definition. The subcircuit is called `amp` and is connected between the nodes `in`, `out` and `vdd`.

```
c30 in 31 47u
r30 31 33 390
r31 vdd 33 50k
r32 33 0 15k
r33 vdd out 750
q30 out 33 0 q2n2222
.ends amp
```

The above line indicates the end of the definition of the subcircuit `amp`.

Note

All nodes used within the subcircuit are local nodes, in that they are only referenced within the subcircuit itself and that they do not have to correspond with the names of the nodes outside the subcircuit.

```
r1 1 2 10
r2 4 0 50k
l1 2 3 1.3m
c1 3 0 100n
x1 3 4 9 amp
```

The above line instantiates the subcircuit `x1` of type `amp` connected between the nodes `3`, `4`, and `9`. As explained earlier, these nodes correspond to the nodes `in`, `out` and `vdd` within the subcircuit.

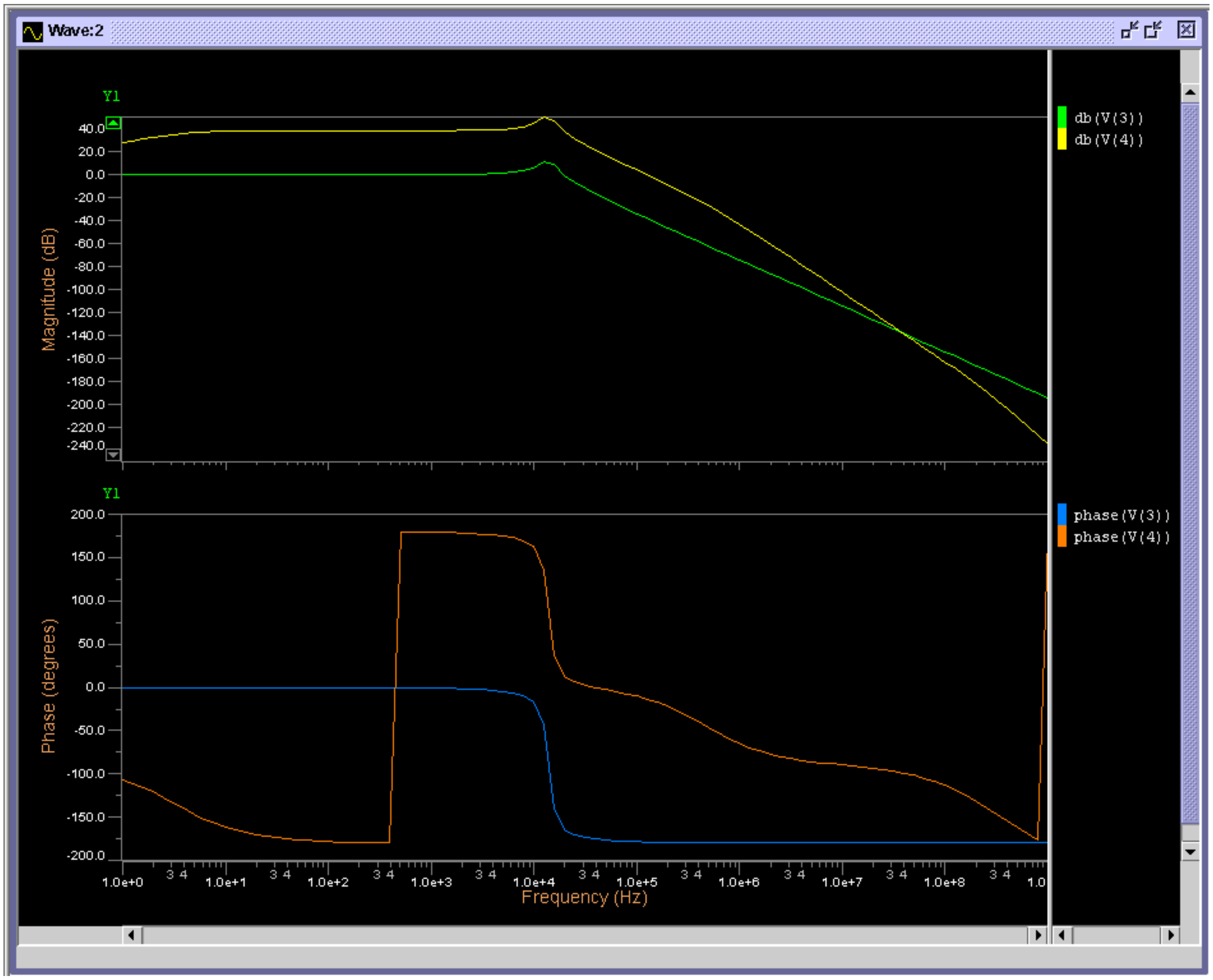
```
vb 9 0 5
v1 1 0 ac 1
```

The above lines define the voltage sources in the circuit. An AC voltage source `v1` connected between the nodes `1` and `0` of value 1 V and a DC voltage source between the nodes `9` and `0` of value 5 V.

```
.ac dec 10 1 1g
.plot ac vdb(3) vdb(4) (-250,50)
.plot ac vp(3) vp(4) (-200,200)
.end
```

Simulation Results

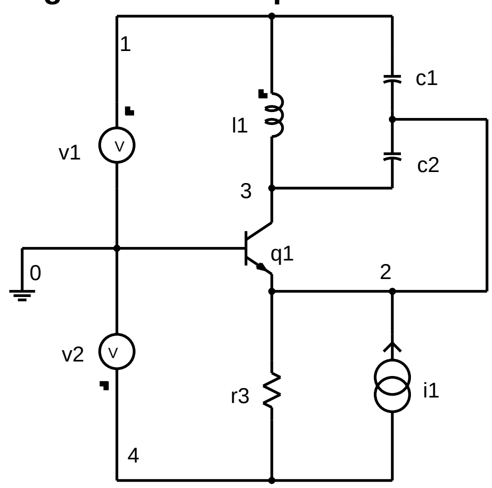
Figure 24-9. Tutorial #4—Simulation Results



Tutorial #5—Colpitts Oscillator

This tutorial deals with the transient analysis of a simple oscillator circuit. It also illustrates making use of model library files found elsewhere in the system environment. The complete netlist can be found in the file *colpitts.cir*.

Figure 24-10. Colpitts Oscillator



Summary of Eldo Commands used in this Tutorial **.MODEL**—Model definition **.OPTION**—Simulator configuration **.PLOT**—Plot simulator results **.TRAN**—Transient analysis

Complete Netlist

.MODEL definition in the library file: `NBJT_LIB`

```
.model ts2 npn
+ bf=10 br=1 xtb=3 is=10f eg=1.11 rb=100
+ rc=10 vaf=50 tr=6n mjc=0.75 mje=0.33 vje=0.75
```

Main Eldo Netlist

```
Colpitts Oscillator
l1 1 3 5u
c1 1 2 2n
c2 2 3 100p
r3 2 4 2200
q1 3 0 2 ts2
v1 1 0 5
v2 4 0 -5
i1 2 4 pulse(0 10u 0 5n 5n 25n 50n)
```

```
.model lib  nbjt_lib ts2
.option eps=1.0e-6
.tran  1u 12u
.plot tran  v(1,3) (10,-10)
.end
```

Netlist Explanation

.MODEL definition in the library file NBJT_LIB:

```
.model ts2 npn
+ bf=10 br=1 xtb=3 is=10f eg=1.11 rb=100
+ rc=10 vaf=50 tr=6n mjc=0.75 mje=0.33 vje=0.75
```

The above lines describe the electrical parameters of the **npn** transistor model ts2.

> For a detailed description of each of these parameters, refer to the Device Models chapter.

Main Eldo Netlist

```
l1 1 3 5u
c1 1 2 2n
c2 2 3 100p
r3 2 4 2200
q1 3 0 2 ts2
```

The above line defines a transistor **q**1 between nodes 3, 0 and 2 with electrical parameters defined by the model ts2.

```
v1 1 0 5
v2 4 0 -5
```

The above lines define the voltage sources in the circuit. A DC voltage source **v**1 connected between the nodes 1 and 0 of value 5V and also a DC voltage source between the nodes 4 and 0 of value –5V.

```
i1 2 4 pulse(0 10u 0 5n 5n 25n 50n)
```

This line defines a time dependent pulse function between nodes 2 and 4 describing the following signal:

0 A at 0s (delay time is 0s)
0 A to 10µA in a rise time of 5ns
10µA from 5 to 30ns (pulse width is 25ns)
10µA to 0A in a fall time of 5ns
0 A at 35ns
Cycle repeats starting from 50ns.

```
.MODEL LIB NBJT_LIB TS2
```

The above line indicates that the electrical parameters of the model `TS2` are defined in the library file `NBJT_LIB` as shown previously.

```
.option eps=1.0e-6
.tran 1u 12u
```

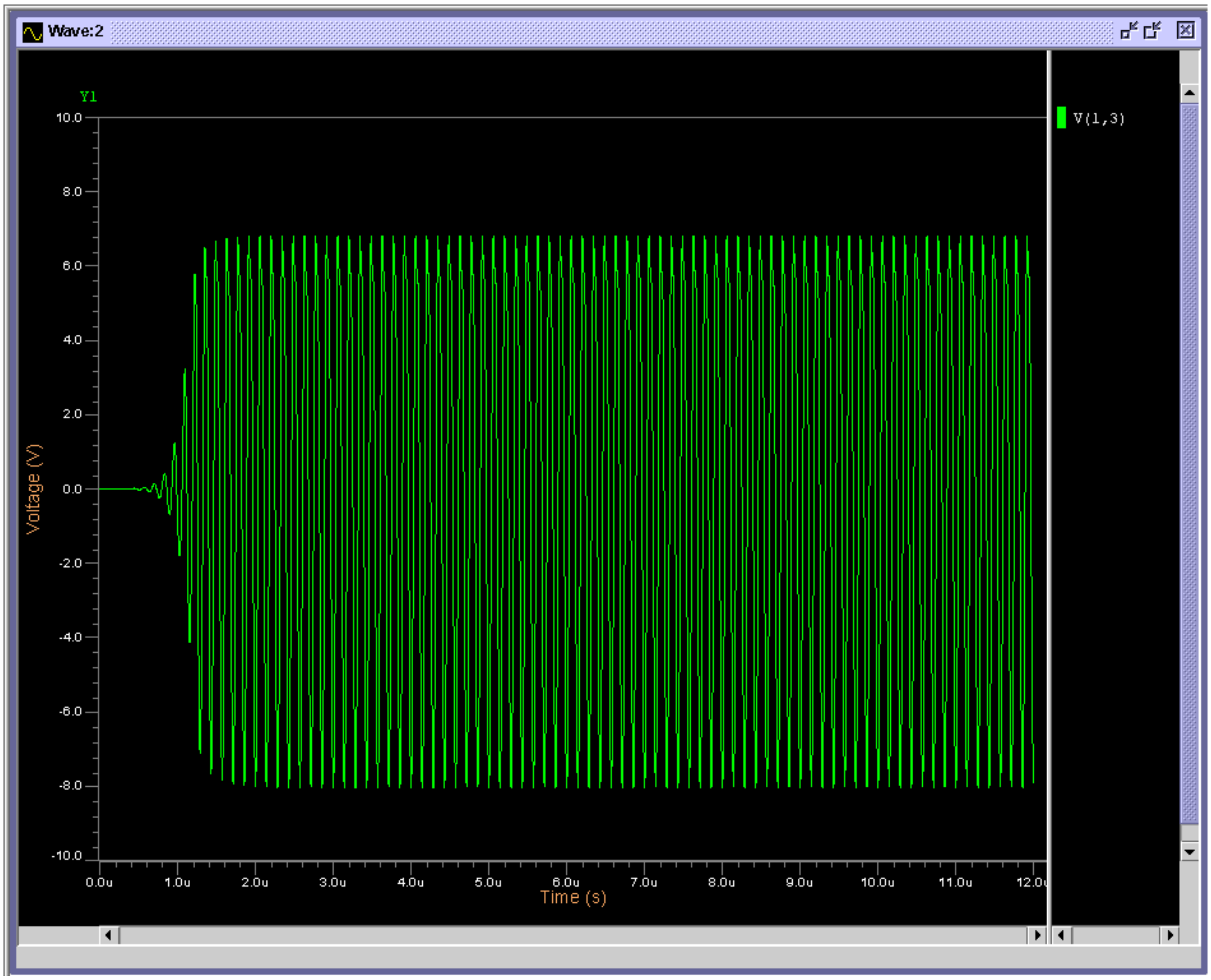
The above specifies a transient analysis is to be performed lasting 12µs with a plotting increment of 1µs.

```
.plot tran v(1,3) (10,-10)
```

The above line specifies a voltage/time plot to be performed of the voltage difference between the nodes `1` and `3` between the limits ±10V.

Simulation Results

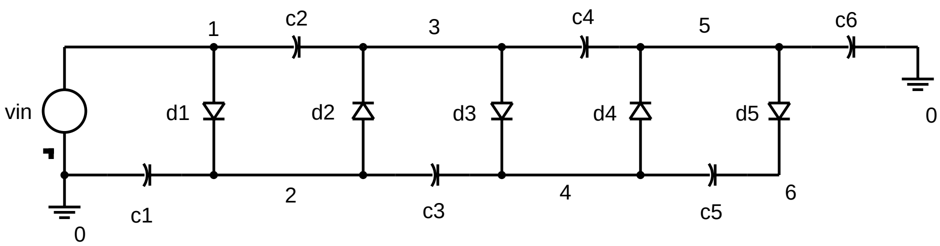
Figure 24-11. Tutorial #5—Simulation Results



Tutorial #6—High Voltage Cascade

This tutorial deals with the transient analysis of a high voltage cascade circuit. The complete netlist can be found in the file *hv_cascade.cir*.

Figure 24-12. High Voltage Cascade



Summary of Eldo Commands used in this Tutorial

.MODEL—Model definition
.OPTION—Simulator configuration
.PLOT—Plot simulator results
.TRAN—Transient analysis
.PARAM—Global parameter setting

Complete Netlist

```
high voltage cascade
.model dl1001 d rs=10 vj=0.8
d1 1 2 dl1001
d2 2 3 dl1001
d3 3 4 dl1001
d4 4 5 dl1001
d5 5 6 dl1001
c1 2 0 cap1
c2 1 3 cap1
c3 2 4 cap1
c4 3 5 cap1
c5 4 6 cap1
c6 5 0 cap2
vin 1 0 sin(0 2500 50k 0 0)
.param cap1=1n cap2=10n
.option reltol=0.01 vmax=100000 vmin=-100000
.tran 0.5m 5m
.plot tran v(5)
.plot tran v(1)
.end
```

Netlist Explanation

```
high voltage cascade
.model dl1001 d rs=10 vj=0.8
```

The above line describes the electrical parameters of the diode model `dl1001`.

For a detailed description of each of these parameters, refer to the Device Models chapter.

```
d1 1 2 dl1001
d2 2 3 dl1001
d3 3 4 dl1001
d4 4 5 dl1001
d5 5 6 dl1001
c1 2 0 cap1
c2 1 3 cap1
c3 2 4 cap1
c4 3 5 cap1
c5 4 6 cap1
c6 5 0 cap2
vin 0 1 sin(0 2500 50k 0 0)
```

The above line defines a sinusoidal function between the nodes `1` and `0`, with a starting amplitude of 2500 V and a frequency of 50 kHz.

```
.param cap1=1n cap2=10n
```

The above line defines the global parameters `cap1` and `cap2` that are used in the capacitor and diode definitions to be 1 nF and 10 nF respectively.

```
.option reltol=0.01 vmax=100000 vmin=-100000
```

The above line sets the relative accuracy to a value of 0.01 and output voltage limits between 100,000 and –100,000 V due to the high voltage levels present in this circuit.

```
.tran 0.5m 5m
```

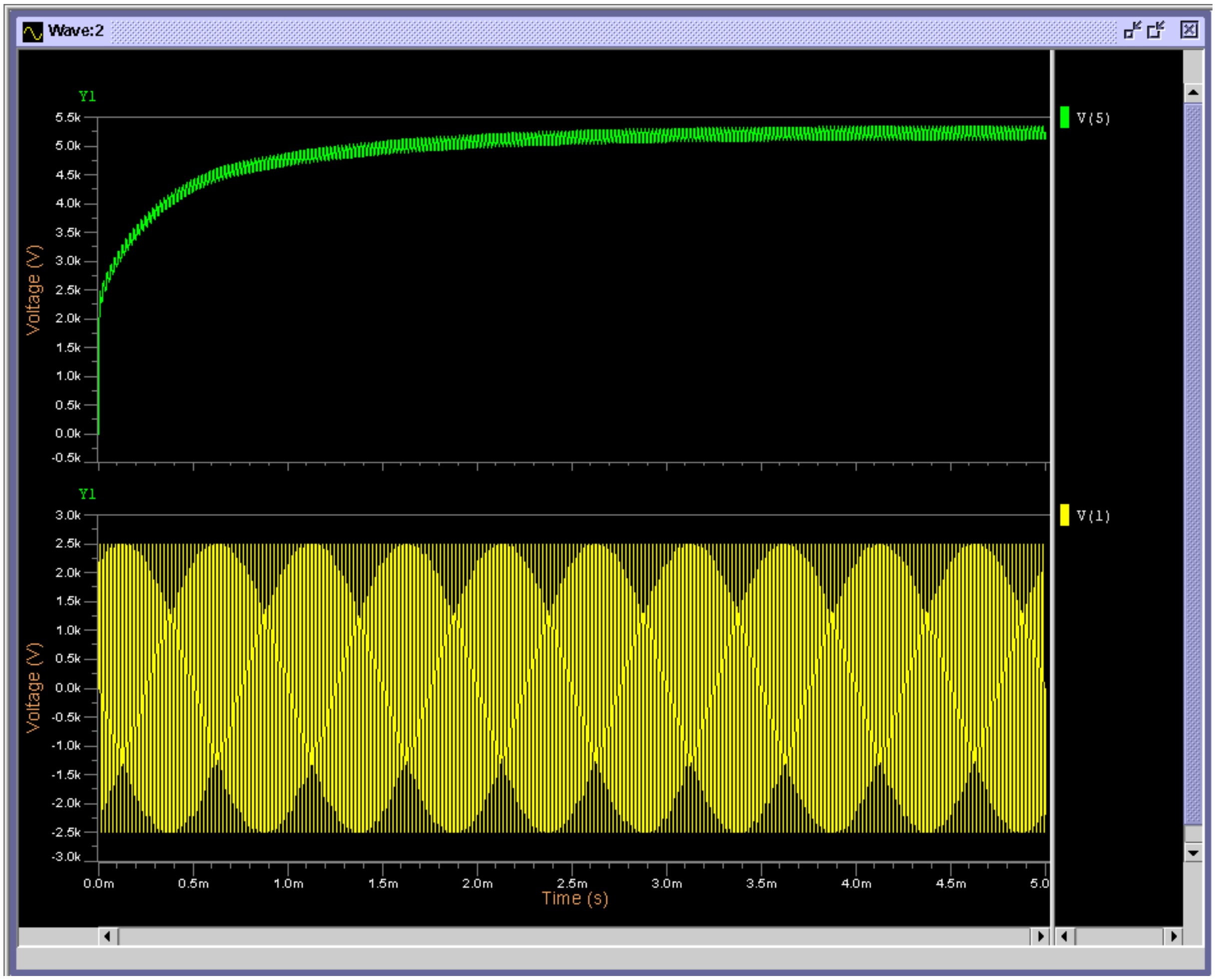
The above line specifies that a transient analysis should be performed on the circuit lasting 5 ms with a plotting increment for the line printer of 0.5 ms.

```
.plot tran v(5)
.plot tran v(1)
```

The above lines specify voltage/time plots to be performed on separate graphs of the voltages at nodes `5` and `1`.

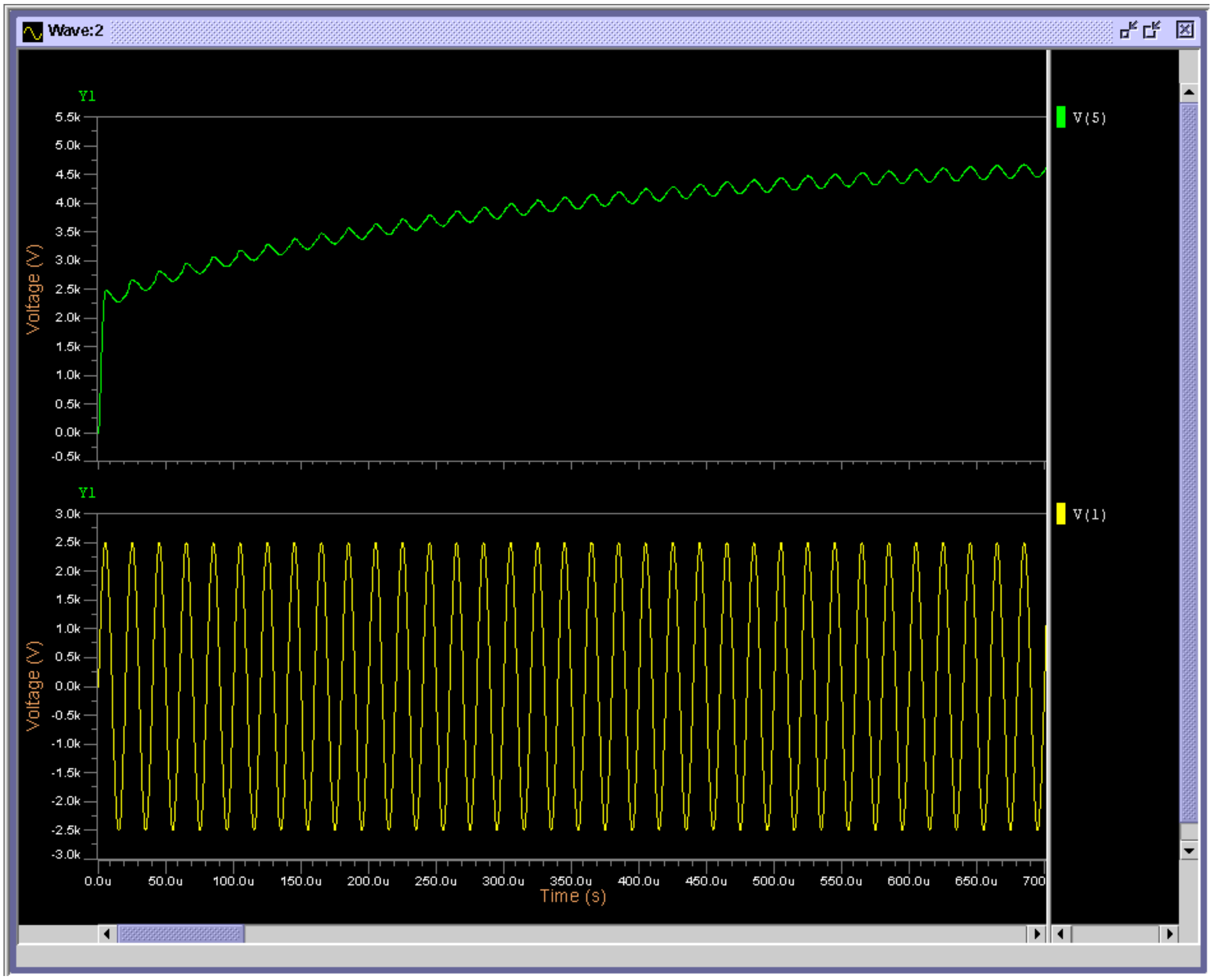
Simulation Results—1

Figure 24-13. Tutorial #6—Simulation Results—1



Simulation Results—2

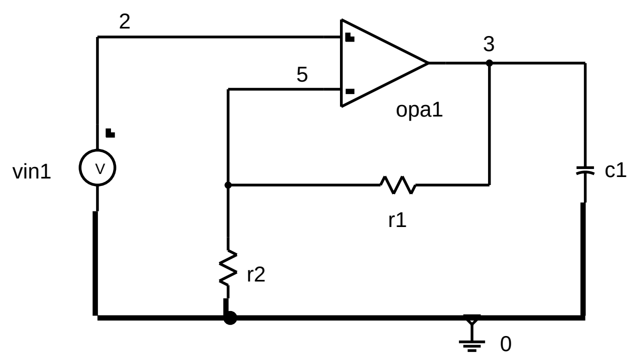
Figure 24-14. Tutorial #6—Simulation Results—2



Tutorial #7—Non-inverting Amplifier

This tutorial deals with Monte Carlo analysis on a non-inverting amplifier circuit. A specified number of simulation runs are carried out to see the effect on the circuit of changing component values within a specified range during each simulation run. Upper and lower limits of the outputs are then displayed by Eldo at the end of the simulation. The complete netlist can be found in the file *noninvert_amp.cir*.

Figure 24-15. Non-inverting Amplifier



Summary of Eldo Commands used in this Tutorial
.AC—AC analysis **.MC**—Monte Carlo analysis **.MODEL**—Model definition **.OPTION**—Simulator configuration **.PLOT**—Plot simulator results

Complete Netlist

```
Non-Inverting Amplifier
.model modres res lot=50% dev=60%
r1 5 3 modres 100k
r2 5 0 modres 1k
c1 3 0 1p
yopa opamp2 pin : 2 5 3 0  param: p1=1e3 p2=5e8 gain=5000
vin1 2 0 ac
.ac dec 30 1.0 100meg
.option  eps=1.0e-6
.mc 7 vdb(3)
.print ac vdb(3)
.plot ac vdb(3)(-40,60)
.plot ac vp(3) (0,-90)
.end
```

Netlist Explanation

```
Non-inverting amplifier
.model modres res lot=50% dev=60%
```

The above line specifies the Monte Carlo parameter limits for resistor model type `modres`. It indicates that for each Monte Carlo run the resistor values can change together by as much as 50% (**`lot`** tolerance). Additionally, the resistor values are allowed to change independently of each other by as much as 60% (**`dev`** tolerance). As can be seen below, these limits will have a more profound effect on the resistor `r2` than that of `r1`.

```
r1 5 3 modres 100k
r2 5 0 modres 1k
c1 3 0 1p
```

The above lines give the information of the resistors and capacitors that are present in the circuit to be simulated. Listed is the component name, the nodes between which the component is connected and the value of the component.

Note

The resistors are also defined to be of model type `modres`. This is present in order to specify Monte Carlo parameter limits for the resistors during the simulation runs.

```
yopa opamp2 pin: 2 5 3 0  param: p1=1e3 p2=5e8 gain=5000
vin1 2 0 ac
.ac dec 30 1.0 100meg
.option  eps=1.0e-6
.mc 7 vdb(3)
```

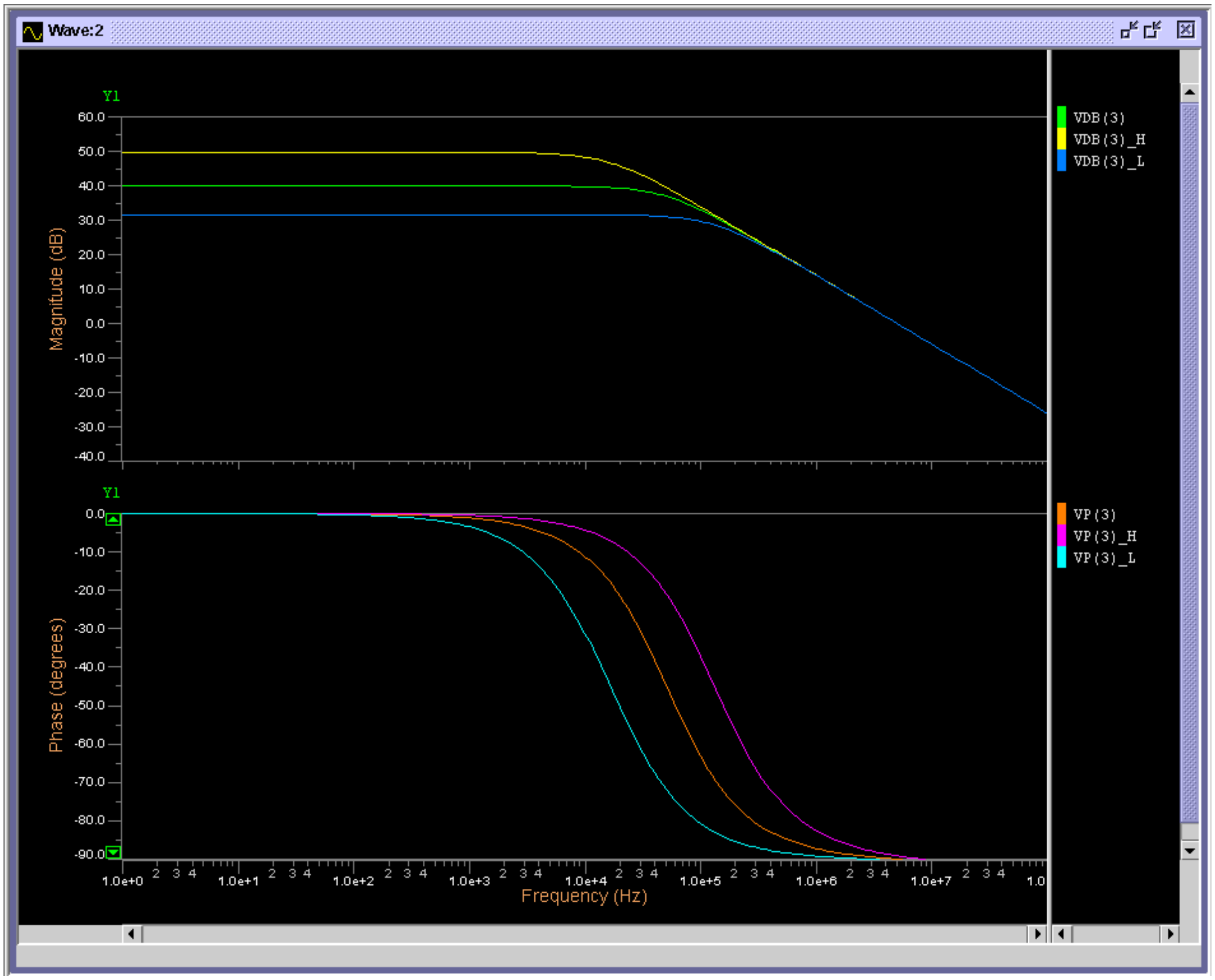
The above line indicates that seven Monte Carlo simulation runs should be carried out on the voltage at node `3`. The plotted output results contain nominal simulation results without any change in the circuit values, together with highest and lowest deviation results over the seven simulation runs.

```
.print ac vdb(3)
.plot ac vdb(3)(-40,60)
.plot ac vp(3) (0,-90)
.end
```

For more information on Monte Carlo analysis, refer to “.MC” on page 706.

Simulation Results

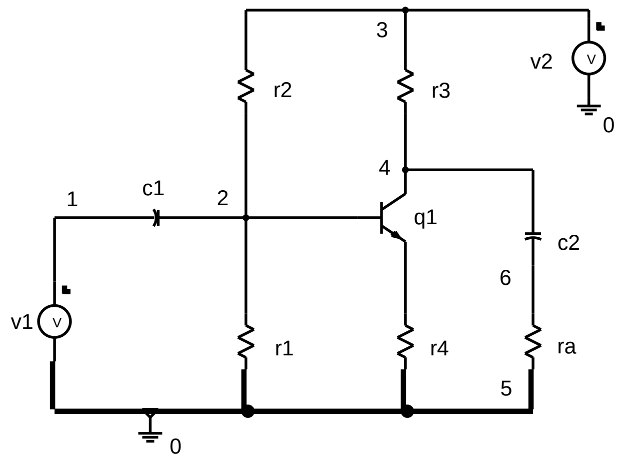
Figure 24-16. Tutorial #7—Simulation Results



Tutorial #8—Bipolar Amplifier

This tutorial deals with a DC sensitivity analysis on the output of a bipolar amplifier circuit. The results show us which DC components have the most effect on the output of the circuit if they were to be changed. The complete netlist can be found in the file *bip_amplifier.cir*.

Figure 24-17. Bipolar Amplifier



Summary of Eldo Commands used in this Tutorial

.MODEL—Model definition
.SENS—Sensitivity analysis

Complete Netlist

```
bipolar amplifier
.model tun1 npn rb=524 irb=0.0 rbm=25 rc=150 re=1.0
+ is=121e-18 eg=1.206 xti=2 xtb=1.538 bf=137 ikf=6.9e-3
+ nf=1 vaf=159 ise=36e-16 ne=1.7 br=0.7 ikr=2.2e-3
+ nr=1 var=10.7 isc=0.0 nc=2 tf=0.6e-9 tr=54.e-9
+ cje=0.2e-12 vje=0.5 mje=0.24 cjc=1.8e-13 vjc=0.5
+ mjc=0.3 xcjc=0.3 cjs=1.3e-12 vjs=0.7 mjs=0.2 fc=0.9
+ itf=40.e-3 vtf=10 xtf=7
r1 2 0 6.8k
r2 3 2 100k
r3 3 4 1.8k
r4 5 0 100
ra 6 0 2.2k
c1 1 2 0.47u
c2 4 6 1u
q1 4 2 5 tun1
v1 1 0 0
v2 3 0 24
.sens v(4)
.end
```

Netlist Explanation

```
bipolar amplifier
.model tun1 npn rb=524 irb=0.0 rbm=25 rc=150 re=1.0
+ is=121e-18 eg=1.206 xti=2 xtb=1.538 bf=137 ikf=6.9e-3
+ nf=1 vaf=159 ise=36e-16 ne=1.7 br=0.7 ikr=2.2e-3
+ nr=1 var=10.7 isc=0.0 nc=2 tf=0.6e-9 tr=54e-9
+ cje=0.2e-12 vje=0.5 mje=0.24 cjc=1.8e-13 vjc=0.5
+ mjc=0.3 xcjc=0.3 cjs=1.3e-12 vjs=0.7 mjs=0.2 fc=0.9
+ itf=40.e-3 vtf=10 xtf=7
r1 2 0 6.8k
r2 3 2 100k
r3 3 4 1.8k
r4 5 0 100
ra 6 0 2.2k
c1 1 2 0.47u
c2 4 6 1u
q1 4 2 5 tun1
v1 1 0 0
v2 3 0 24
.sens v(4)
```

The above line indicates that a DC sensitivity analysis should be performed showing the relative sensitivities that the DC components have on the voltage on node `4`, the output node of the amplifier. The results are listed in the *bip_amplifier.chi* file.

```
.end
```

The results of the sensitivity analysis, found in the *bip_amplifier.chi* file, are listed overleaf.

Simulation Results

Figure 24-18. Tutorial #8—Simulation Results

```
     DC SENSITIVITIES OF OUTPUT V(4)
              ELEMENT       ELEMENT        ELEMENT       NORMALIZED
               NAME          VALUE       SENSITIVITY    SENSITIVITY
                                         (VOLTS/UNIT)  (VOLTS/PERCENT)

               R1          6.800E+03      -1.39E-03     -9.45E-02
               R2          1.000E+05      1.180E-04     1.180E-01
               R3          1.800E+03      -3.90E-03     -7.02E-02
               R4          1.000E+02      3.297E-02     3.297E-02
               RA          2.200E+03      0.000E+00     0.000E+00
               V1          0.000E+00      0.000E+00     0.000E+00
               V2          2.400E+01      4.585E-01     1.100E-01
     Q1
               RB          3.489E+02      3.689E-04     1.287E-03
               RC          1.500E+02      9.155E-05     1.373E-04
               RE          1.000E+00      3.297E-02     3.297E-04
               BF          1.370E+02      -1.82E-02     -2.49E-02
               JLE/ISE     3.600E-15      6.409E+12     2.307E-04
               BR          7.000E-01      1.394E-11     9.759E-14
               JLC/ISC     0.000E+00      0.000E+00     0.000E+00
               JS/IS       1.210E-16      -1.93E+15     -2.34E-03
               NLE         1.700E+00      -2.52E-01     -4.28E-03
               NLC         2.000E+00      0.000E+00     0.000E+00
               JBF/IKF     6.900E-03      -1.43E+02     -9.87E-03
               JBR/IKR     2.200E-03      2.771E-11     6.095E-16
               VBF         1.590E+02      2.163E-03     3.439E-03
               VBR         1.070E+01      -2.60E-02     -2.78E-03
```

Referring to the above results, the *element sensitivity* is the change in the output of interest (in this case the voltage at node `4`) due to a one unit change in the value of the element of interest (for example `r1`) and the *normalized sensitivity* is the change in the output of interest due to a one percent change in the value of the element of interest.

Looking at the sensitivity output in normalized sensitivity (volts/percent), it can be seen that the output at node `4` is most sensitive to the voltage source `v2` and also to the `bf` parameter in `q1`. As a result, variation in these values causes a significant effect on the output voltage.

Please refer to ".SENS" on page 863.

Tutorial #9—SC Low Pass Filter

This example deals with the transient & small signal AC analysis of an SC low pass filter using the Z-domain switched capacitor models. The complete netlist can be found in the file *sc_lowpass.cir*.

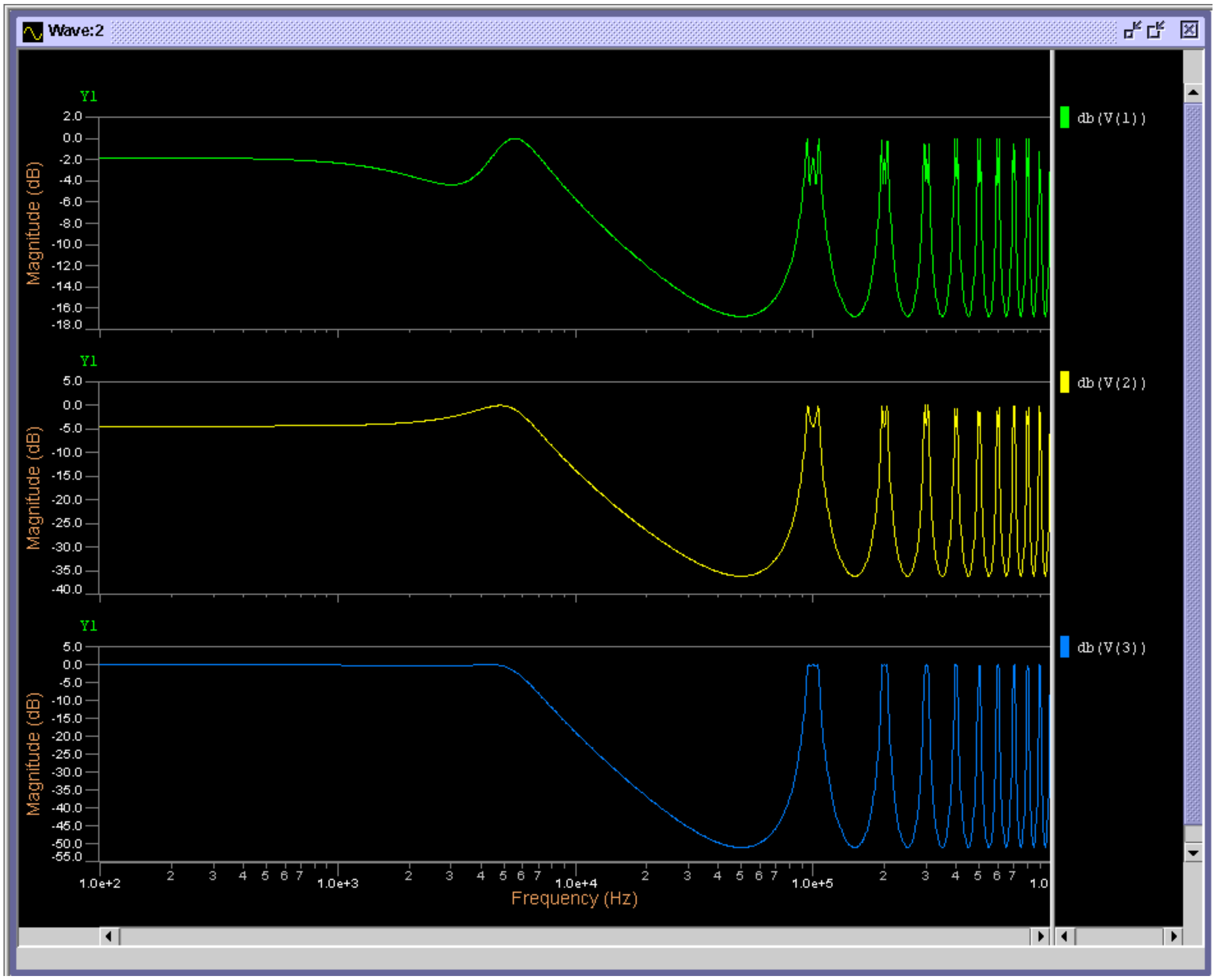
Summary of Eldo Commands used in this Tutorial
.AC—AC analysis **.PLOT**—Plot simulator results **.TRAN**—Transient analysis

Complete Netlist

```
SC_lpfilt.cir
.param ts = 1.000000e-05
.subckt sc_int inp inm out
y1 sc_ideal inp inm out
y2 sc_u inm out param: c=c tp=tp
.ends sc_int
* INTEGRATOR 1
xi01 0 501 1 SC_INT c=5.173415p tp=ts
y002 sc_n pin: 2 0 501 0 param: c=1.347451p tp=ts
y003 sc_n pin: INPUT 0 501 0 param: c=1.623277p tp=ts
y004 sc_n pin: 1 0 501 0 param: c=1.000000p tp=ts
* INTEGRATOR 2
xi05 0 502 2 sc_int c=5.839726p tp=ts
y006 sc_i pin: 1 0 502 0 param: c=1.232076p tp=ts ldi=2
y007 sc_i pin: 3 0 502 0 param: c=1.000000p tp=ts ldi=2
* INTEGRATOR 3
xi08 0 503 3 sc_int c=4.117249p tp=ts
y009 sc_n pin: 2 0 503 0 param: c=1.660161p tp=ts
y010 sc_n pin: 3 0 503 0 param: c=1.000000p tp=ts
.tran 1u 500u
.ac dec 500 100 1meg
.plot tran v(input)
.plot tran v(1)
.plot tran v(2)
.plot tran v(3)
.plot ac vdb(1)
.plot ac vdb(2)
.plot ac vdb(3)
vin input 0 dc 1.0 ac 1.0 pwl(0.0 0.0 10n 1.0)
.end
```

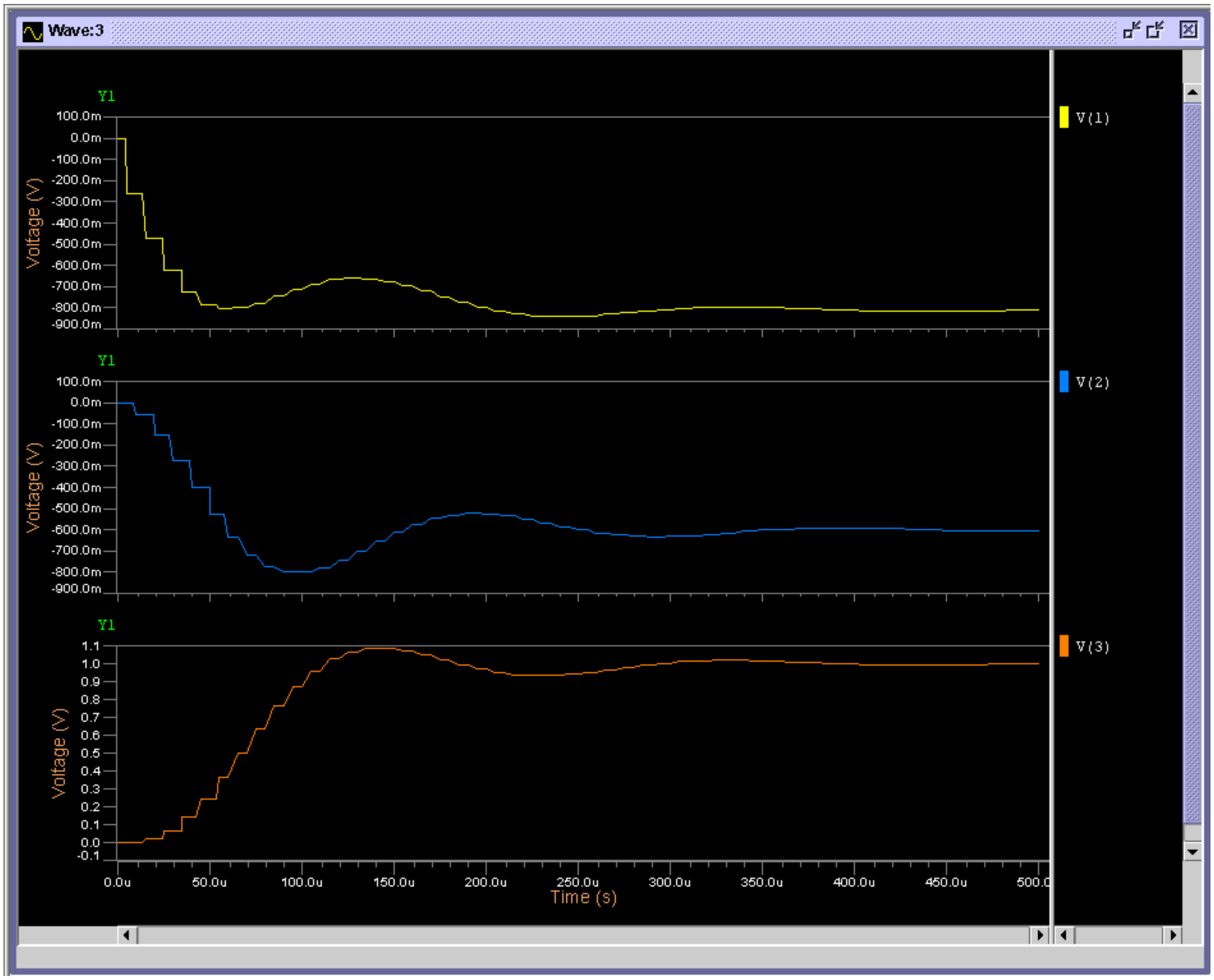
Simulation Results—1

Figure 24-19. Tutorial #9—Simulation Results—1



Simulation Results—2

Figure 24-20. Tutorial #9—Simulation Results—2



Chapter 25
Simulator Compatibility

Introduction

Eldo provides compatibility with different simulators. Eldo can use an external simulator netlist as input, accepting the different constructs and syntax used by such simulators. This chapter shows the compatibility available. It is divided into the following sections:

- HSPICE Compatibility
- TIspice Compatibility
- Spectre Compatibility

HSPICE Compatibility

Eldo provides partial compatibility with HSPICE®[1]. Eldo can use an HSPICE netlist as input. Eldo HSPICE compatibility is invoked as follows:

```
eldo -compat ... cir_file_name
```

or by adding in the netlist the following option (must be set at the top of the design):

```
.OPTION COMPAT
```

The main effect is that it accepts some HSPICE constructs and syntax. This section shows the different effects that this compatibility mode has. It is divided into the following:

- Devices
- Commands
- Options
- Netlist
- Arithmetic Functions and Operators
- Output Format

Additional flexibility is offered. Instead of using `-compat`, the flags `-compmod` and `-compnet` can be set with the following effects:

1. HSPICE is a registered trademarks of Synopsys, Inc.

`-compmod` Triggers only the automatic conversion of models (can alternatively be set with **`.OPTION COMPMOD`**); see Devices.

`-compnet` Causes the netlist to be interpreted as compatible format, but the models themselves are treated as Eldo Spice models. This means it is assumed that models are already Eldo models (for example, to select the BSIM3v3 model the level specified must be 53, not 49); (`-compnet` can alternatively be set with **`.OPTION COMPNET`**).

Note

The `-compnet` and `-compmod` flags apply to the entire netlist, including library files. For example, this implies that when compnet is set, even the statements in the library file, including those of model library files, are interpreted in `-compat` mode.

Note

Flag `-compat` (or **`.OPTION COMPAT`**) is equivalent to setting both `-compmod` and `-compnet` flags (or **`.OPTION COMPMOD`** and **`.OPTION COMPNET`**).

Devices

Eldo Levels

The following Eldo Levels are set:

Table 25-1. MOS Levels with -compat

LEVEL	Model Name (Eldo Level)
2	Eldo2 (Eldo LEVEL 12)
3	Eldo3 (Eldo LEVEL 13)
6	Modified Lattin-Jenkins Grove Model (Eldo LEVEL 16)
8	Enhanced Berkeley LEVEL 2 (Eldo LEVEL 17)
13	Berkeley BSIM1 (Eldo LEVEL 8)
39	Berkeley BSIM2 (Eldo LEVEL 11)
49	Berkeley BSIM3 v3.0 & BSIM3 v3.1 (Eldo LEVEL 53)
50	Philips MOS Model 9 (Eldo LEVEL 59)
54	Berkeley BSIM4.0.0 (Eldo LEVEL 60)
57	Berkeley BSIMSOI3v2 PD (Eldo LEVEL 56, SOIMOD=1)
59	Berkeley BSIMSOI3v2 FD (Eldo LEVEL 56, SOIMOD=3)
68	HiSIM Model (Eldo LEVEL 66)

Table 25-1. MOS Levels with -compat

LEVEL	Model Name (Eldo Level)
70	BSIMSOIv4 Model (Eldo LEVEL 72)

For more information see "MOS Levels with -compat" on page 255.

Table 25-2. BJT Models with -compat

LEVEL	Model Name (Eldo Level)
2	Improved Berkeley Model (Eldo LEVEL 5)
3	STMicroelectronics LEVEL 1 (Eldo LEVEL 2)
4	VBIC v1.2 (Eldo LEVEL 8)
	(**VERSION**=1.15) VBIC v1.1.5 (Eldo LEVEL 8)
6	Philips Mextram 503.2 Model (Eldo LEVEL 4)
8	HICUM Model (Eldo LEVEL 9)

For more information, see "BJT Models with -compat" on page 236.

Table 25-3. Diode Models with -compat

LEVEL	Model Name (Eldo Level)
2	Fowler-Nordheim (Eldo LEVEL 3)
3	Berkeley Level 1 (Eldo LEVEL 1), by default SCALEV is set to 3
4	JUNCAP Diode Model (Eldo LEVEL 8), by default DIOLEV is set to 9

For more information, see "Using -compat with Diodes" on page 229.

Table 25-4. Resistor Level with -compat

LEVEL	Model Name (Eldo Level)
1	RC Wire (Eldo LEVEL 3)

For more information, see "RC Wire Model Syntax" on page 150.

PNP and NPN devices

The `-compat` flag affects the default type for PNP and NPN devices. **`SUBS`** defines the type of BJT. By default, NPN and PNP are vertical:

If **`SUBS`** is -1, BJT is lateral.
If **`SUBS`** is 1, BJT is vertical.

If `-compat` is set, NPN are vertical, PNP are lateral.

For more information, see "BJT Model Syntax" on page 235.

Group Delay

`VT` is equivalent to **`VGD`**, and **`IT`** is equivalent to **`IGD`** (Group Delay on voltage or current). For more information, see "`VXxx(devname.posi)`" on page 800.

Node GROUND/GND/GND!

Node **`GROUND`**, **`GND`** and **`GND!`** are assumed to be the global node 0.

Value "x"

"x" is equivalent to "meg", i.e. when the `-compat` flag is set:

```
R1 1 0 1x
```

is equivalent to:

```
R1 1 0 1meg
```

Otherwise, `1x` corresponds to 1.0.

Commands

- **`.ALTER`**

  This command is cumulative. For example:

```
r1 1 0 1
r2 2 0 1
.alter 1
r1 1 0 2
.alter 2
r2 2 0 2
```

The second "alter" simulation will be done with both `r1` set to 2 (inheritance from later number 1), and `r2` set to 2. For more information see ".ALTER" on page 549.

Option **COMPAT** can be placed anywhere in the netlist; it is not order-dependent. However, if the netlist contains **.ALTER** statements, and the option is placed in the **.ALTER** block then it will not be taken into account in the nominal run, only the **.ALTER** run. Then it will remain active for any further **.ALTER** runs. For example, if the netlist contains:

```
title
netlist
.alter 1
.option compat
.alter 2
...
.end
v1 1 0 1
r1 1 0 1
.dc
.end
```

Here, the nominal run will not recognize the **COMPAT** option, the first **.ALTER** will, and the second **.ALTER** will also. The statements between the last two **.END** statements, which correspond to the simulation of yet another circuit, will still consider **COMPAT** as active. Once activated, compat cannot be deactivated.

- **.DC**

  When only a DC analysis is specified in conjunction with the `-compat` flag, by default the initial transient value will be used. For more information see ".DC" on page 588.

- **.HDL**

  The syntax is different with the `-compat` flag:

  .HDL `"filename" [module_name] [module_alias]`

  See also ".HDL" on page 678.

- **.LIB**

  .LIB behaves as **.INCLUDE**. For more information see ".LIB" on page 692.

- **.PLOT**/**.PRINT**

  The sign convention of **I**`x` is that the current is positive when it enters the device by pin `x`. However, when `-compat` is set, then the following apply:
  For R/L/C/E/F/G/H/I/V/D devices, **I**`2` returns the same value as **I**`1`.
  For M/B/J devices, **I**`3` is positive when current leaves the object by pin number 3. For more information see ".PLOT" on page 791 and ".PRINT" on page 830.

- **.PROBE**

  The Eldo **.PROBE** card is emulated, i.e. all node voltages are dumped in the binary output file (*.wdb*). In order to have only those items specified in **.PLOT**/**.PROBE** to be dumped in the output file, add **.OPTION PROBE**. For more information see ".PROBE" on page 838.

- **.SUBCKT**

  Usage of **GLOBAL** node names in the **.SUBCKT** definition node list. Eldo will check the node names in the subcircuit list definition prior to the global node list. For more information see ".SUBCKT" on page 898.

- **.TEMP**

  The model parameter **TREF** can be specified. This is equivalent to the parameter **TNOM** when the **.TEMP** command is used to apply temperature effects. When **TREF** is specified, **TNOM** will be ignored.

- **.TRAN**

  If the **.TRAN** command has four parameters, for example:

  ```
  .TRAN tprint tstop tstart hmax
  ```

  - if value4 (hmax) < value2 (tstop), it is treated as in Eldo standard mode:

    ```
    .TRAN tprint tstop tstart hmax
    ```

  - if value4 (hmax) ≥ value2 (tstop), it is treated as a list of INCRn Tn values:

    ```
    .TRAN INCR1 T1 [{INCRn Tn}] [TSTART=val] [UIC]
    ```

  For more information see ".TRAN" on page 911.

Options

- **NOINIT**

  Set for cases where **UIC** is set on the **.TRAN** card. For more information, see "UIC" on page 912.

- **ACM**

  This option is automatically set in compat mode. For more information, see "ACM" on page 983.

- **ICDC** and **ICDEV**

  These options are automatically set in compat mode. This means that IC specifications given on devices are taken into account for the DC which is performed prior to TRAN. When -compat is not set, and neither are the **ICDC** and **ICDEV** options, then IC specifications given on devices are ignored, unless the **UIC** keyword is specified on the **.TRAN** card. For more information, see "ICDC and ICDEV" on page 978.

- **LICN**

  This option is automatically set in compat mode. Causes the last initial condition (**.IC**) specification to have precedence over previous IC specifications.

- **(NO)KWSCALE**

  This option is automatically set in compat mode. Therefore, **SCALE** is not considered as a keyword and can be used as a parameter. For more information, see "(NO)KWSCALE" on page 991.

- **ACOUT**=VAL

  ACOUT defaults to 1.

  1

  Vx(a,b) or **Ix**(a,b) are computed from the complex value **Vx**(a,0)-**Vx**(b,0) or from **Ix**(a,0)-**Ix**(b,0). For more information, see "ACOUT=VAL" on page 997.

- **PROBE**

  .PROBE command is emulated, i.e. all node voltages are dumped in the binary output file (*.wdb*). In order to have only those items specified in **.PLOT**/**.PROBE** to be dumped in the output file, add **.OPTION PROBE**. For more information, see "PROBE" on page 1013.

- **LIBINC**

  This option is automatically set in compat mode. It specifies that all the libraries (.LIB) should be included without filtering the objects (model, card or subcircuit) that are not used in the specific netlist. For more information, see "LIBINC" on page 966.

- **SLASHCONT**

  This option is automatically set in compat mode. It allows a single backslash to be used for the continuation of the line. Example:

  ```
  .OPTION SLASHCONT
  R1 1 2 \\
  3k
  R2 1 2 1k
  ```

  R1 will be set to 3K. It is possible to disable this option by using **.OPTION NOBSLASHCONT**.

- **PARHIER**='local'|'global'

  Default is set to global. Global "parent" values have precedence. Example:

  ```
  .SUBCKT R1 1 2
  R1 1 2 a
  .ends
  X1 in out R1 a=2
  .param a=3
  ```

  When **parhier** is local, X1.R1 will be 2, while when **parhier** is global, X1.R1 is 3. For more information, see the PARHIER description in the Simulator and Control Options chapter.

- **GENK**<=val>

  This option is automatically set in compat mode. It forces Eldo to generate 2nd order mutual inductors. It is used together with the **KLIM** option. For more information, see "GENK[=VAL]" on page 986.

- **TNOM**

  TNOM defaults to 25˚C instead of 27˚C. For more information see "Temperature Handling" on page 88.

- **DEFPTNOM**

  This option is automatically set in compat mode. Allows a parameter to be defined with the name TNOM. In such a case, this value will be used inside parameter expressions, instead of the default **TNOM** or the value set using option **TNOM**=val.

- **MODWLDOT**

  This option is automatically set in compat mode. It is used for binned models. For more information, see "MODWLDOT" on page 988.

- **NOELDOSWITCH**

  When specified with the -compat flag/option, it informs Eldo that devices beginning with an *S* are not switches (Eldo default) but S-parameter block instantiations.

- **QUOTREL**

  This option is automatically set in compat mode. For more information, see "QUOTREL" on page 959.

- **NOELDOLOGIC**

  This option is automatically set in compat mode. For more information, see "NOELDOLOGIC" on page 955.

- **ACDERFUNC**

  This option is automatically *unset* in compat mode. For more information, see "ACDERFUNC" on page 983.

Options Only Available in compat Mode

The following options are only available in compat mode:

- **ALTERELDO**

  Changes the way the **.ALTER** re-run feature works in compat mode. In compat mode, for alter index 'n', Eldo revisits the 'n-1' alter looking for substitution. In default Eldo mode, for alter index 'n', Eldo only deals with the nominal and the alter 'n'; it ignores the 'n-1' alter. In other words, in compat mode, alters are cumulative, but not in Eldo default mode. Specify **ALTERELDO** to activate the default Eldo mode behavior when in compat mode.

- **CKDCPATH**

  Forces Eldo to issue a warning instead of an error when a dangling node (no DC path to ground) is encountered on a current source. The source is then disabled and the node connected to ground. This option is only used in compat mode.

- **COMPEXUP**

  Forces Eldo to keep the name of the extraction results (from **.EXTRACT** and **.MEAS** statements) in uppercase. This option is only used in compat mode.

- **NOBSLASHCONT**

  Disables option **SLASHCONT** which is set by default in compat mode.

Netlist

Comment character

Dollar `$` is used as the comment character in the netlist line instead of `!`.

The comment character is usually considered as such only if the preceding character is a blank space or if the comment is at the beginning of a line. In `-compat` mode with the comment character `$`, then the above rule still applies; additionally, the `$` character is a comment character if:

- the preceding character is not a white space, *and*
- the first character of the string is ‘, ’, “, a number, or a period (.).

If you want to reference an environment variable using the `$` character, for example to specify a file path, surround the path and filename in quotes.

Examples:

- Circumstances where `$` is considered as a comment:
  - `$` is considered a comment because the string begins with the digit 0:

    ```
    M1... l=0.1u$
    ```
  - `$` is considered a comment because the preceding character is a blank space:

    ```
    M1... l=0.1u $
    .INCLUDE $HERE/myfile.txt
    ```
  - `$` is considered a comment because the first character is a number:

    ```
    M1 1$ d g s b nmos w=1u l=3
    ```
- Circumstances where `$` is *not* considered as a comment:

  ```
  M1 a1$ d g s b...
  M2 ‘1$’ d g s b ...
  .INCLUDE "$HERE/myfile.txt"
  ```

Parameters

Multiple affectation of parameters which can be overwritten sequentially is allowed.

In model instantiations, Eldo will check whether there is a **.model** with the same name as the string after the nodes in a model declaration. If not, it will look for a parameter name.

In **.PARAM** statements, **LOT** and **DEV** are not considered as keywords, however **LOT/GAUSS**, **DEV/GAUSS**, **LOT/**<distrib_name>, and **DEV/**<distrib_name> are.

Quotes

Double quotes are considered as single quotes. (In standard Eldo mode, double quotes are used to specify a parameter string.) Use option **QUOTSTR** to consider double quotes as a parameter string delimiter.

Order of analyses

Analyses will be performed in the order specified in the netlist. The order in which analyses will be performed can then differ from the original Eldo order, and there can be multiple analyses of the same type to be run sequentially. In addition, **.PRINT**/**.PLOT**/**.EXTRACT**/**.MEAS** commands will be seen only by the preceding command. Example:

```
.tran 1n 10n
.print tran v(1)
.tran 1n 50n
.print tran v(2)
.end
```

With the -compat flag set, the first **.tran** command will force Eldo to do a transient simulation between 0 and 10n. Only v(1) will be displayed. The second **.tran** command will force Eldo to do a transient simulation between 0 and 50n. Only v(2) will be displayed for that simulation.

Order of analyses—AC in the middle of a .TRAN

AC in the middle of transient is handled with respect to the compat mode ordering rule (as described above). Whenever a **.AC** command, a **.TRAN** command, and a **.OP** command containing time specifications are specified in the *.cir* file, then the results will be:

- **.ac** followed by **.tran**

  AC will be performed during the transient because **.ac** appears before **.tran**

- **.tran** followed by **.ac**

  AC will be performed after the transient, and not inside it. A warning is issued:

```
Warning 1484: AC analysis will not be performed during transient because
of the commands ordering.
```

- `.tran` followed by `.ac` followed by another `.tran`

  The first transient is performed, then AC + transient as in the first case

Arithmetic Functions and Operators

When the `-compat` flag is active, the following arithmetic function/operator rules apply:

```
log(x) = sign(x) * log(abs(x))
log10(x) = sign(x) * log10(abs(x))
db(x) = sign(x) * 20.0*log10*abs(x))
```

`sqrt(x)` is `-sqrt(abs(x))` if `x` is negative.
`x**n` is computed as `x**n` if `x` is positive, `-(abs(x)**n)` if `x` is negative, and 0 if `x` is 0.
The power operator (^) has highest precedence (same as standard Eldo); prior to v6.3_2 it had lower precedence in -compat mode than the multiplication and division operators.

Note
In Eldo standard mode:
`sqrt(x)` returns an error if `x` is negative.
`x**n` is computed as `exp(n*log(x))` if `x` is strictly positive, 0 otherwise.

Tip: For more information, see “Arithmetic Functions” on page 78.

EVAL keyword

When the `-compat` flag is active, the function keyword `EVAL` is not required in conditional expressions. For example:

```
r1 1 2 '(p1 > p2 ?  p2: p1)'
```

Will be the equivalent of:

```
r1 1 2 'eval(p1 > p2 ?  p2: p1)'
```

Output Format

The following types of output files are generated when the `-compat` flag is active:

- If option **POST**=1 or **POST**=2, Eldo generates only *.tr0* output files, no *.wdb* file generated.
- Without specifying option **POST**, Eldo generates only *.wdb* output files.

TIspice Compatibility

Eldo provides partial compatibility with TIspice version 3.40. TIspice is developed by Texas Instruments Incorporated. This section describes some specific TIspice syntax that is compatible inside Eldo.

Eldo TIspice compatibility is invoked as follows:

```
eldo -tispice ... cir_file_name
```

or by adding in the netlist the following option:

```
.OPTION TISPICE
```

This section shows the different effects that this compatibility mode has.

Devices

Model mapping

The following Eldo Levels are set:

Table 25-5. MOS Levels in TIspice Compatibility Mode

LEVEL	Model Name (Eldo Level)
4	Berkeley BSIM1 (Eldo LEVEL 8)
8 or 49	Berkeley BSIM3 v3.0 & BSIM3 v3.1 (Eldo LEVEL 53)
9	Berkeley BSIMSOI3v3.1.1 PD (Eldo LEVEL 56, SOIMOD=0)
14 or 54	Berkeley BSIM4.0.0 (Eldo LEVEL 60)

Table 25-6. BJT Models in TIspice Compatibility Mode

LEVEL	Model Name (Eldo Level)
2	VBIC v1.2 (Eldo LEVEL 8)
	(**VERSION**=1.15) VBIC v1.1.5 (Eldo LEVEL 8)
504	Philips Mextram 504 Model (Eldo LEVEL 22)

Resistor and capacitor model syntax

.MODEL R or C: SCALE equivalent to Eldo R or C

.MODEL R and no level => level = 5

.MODEL C and no level => level = 5

Resistor model level 5

This is the default resistor model when in Eldo TIspice compatibility mode. L and W are instance parameters. Model parameters:

RSH	Sheet resistivity. Unit is Ohm/square. This value is mandatory.
LR	The reduction in length from side etching. NARROW overrides this parameter. Default is 0.
NARROW	The narrowing of the resistor due to side etching. The units for this are meters, and the default value is 0.
WR	The reduction in width from side etching. NARROW overrides this parameter. Default is 0.
SCALE	The scaling factor (on top of all other calculations) used to multiply the value of the resistance before analysis. The default value is 1.

The R value is computed as:

$$R = RSH \times (L - LR)/(W - WR)$$

If neither L or W are specified, then it is assumed that the R value is specified on the instance card: then only model parameter SCALE will be considered.

Capacitor model level 5

This is the default capacitor model when in Eldo TIspice compatibility mode. Model parameters:

cj	The junction bottom capacitance for semiconductor capacitors. Units for this are farads/square meter. This parameter must be specified for a semiconductor capacitor.
cjsw	The sidewall junction capacitance in farads/square meter for a semiconductor capacitor. This is also a required model parameter for a semiconductor capacitor.
defw	The default width for a semiconductor capacitor. A W=width statement on capacitors overrides this value. The units for this are meters, and the default value is 1E-6.
narrow	The narrowing of the capacitor due to side etching in the case of semiconductor capacitors. The units for this parameter are meters, and the default value is 0.
scale	The scaling factor (on top of all other calculations) used to multiply the value of the capacitance before analysis. The default value is 1.

Usage of these model parameters depends on how the device has been instantiated:

- if W or L specified:

  C= CJ $\times$ (L-NARROW) $\times$ (W-NARROW) + 2 $\times$ CJSW $\times$ (L+W-2$\times$NARROW)

- if AREA or PERI specified:

  C = CJ $\times$ (AREA - NARROW $\times$ PERIMETER / 2 + NARROW^2) + CJSW $\times$ (PERIMETER - 4 $\times$ NARROW)

- None specified: C is assumed to be specified on the instance card.

Resistor, self inductor and capacitor

The model name, if any, is placed after the nominal value for a non-geometric model, and before any parameter for a geometric model:

```
R1 a b value MODEL TC1 = 1
R1 a b MODEL L = ...
```

In the case of a non-geometric model, the 5th token can be a special parameter name rather than a model name. If the 1st letter of the 5th argument is a P, then Eldo will look for a parameter rather than a model. The value of that parameter is a multiplier of the device. For example:

```
R1 1 2 1k pr
.param pr = 2
```

The actual value of R1 will be 2k.

MOS length and width

Length and width of MOS are supposed to be given in Microns. Same for PD and PS. AD and AS are also expected in Microns^2. The default in Eldo is meters.

Commands

.PARAM command

The '=' is not mandatory on the **`.PARAM`** but if the '=' is omitted, there can be only one parameter defined per **`.PARAM`** statement.

.LIB command

If dns=/path on a **`.LIB`** statement is specified, the dns= is ignored. This was old TIspice syntax.

.TRAN command extension

The full TIspice **`.TRAN`** syntax is accepted.

TMIN sets hmin

TSHIFT sets time at which all independence sources begins their evaluation

TPUNCH creates a save file for each tpunch timepoint

WRINIT RDINIT RDFORCE WRFINAL
handles init filenames

The following options are accepted but ignored: SAVE, CHECKSTOPDELAY, TRNOISE, INITERROR.

.INCLUDE/.LIB commands

These commands accept the syntax SECTION = <>, which gives the corner name of the library.

When **NOPRINT** is specified the library netlist is not dumped in the output (*.chi*) file.

.RERUN command

Same as the Eldo **.ALTER** command except that a **.END** statement is required between each **.RERUN**.

.PRINT/.PLOT command extensions

General form:

```
.print  analyse_type  signal_list
.plot   analyse_type  signal list
```

signal_list is the collection of output variables. TIspice output variables can be specified in one of five ways:

- using a full name format
- using a partial name format
- using regular expression format
- specifying depth of hierarchy
- using a mathematical-expression format

Currently only partial name and full name formats are supported in Eldo. An example of partial name format is:

```
.print ac all(v i) except(i(x1.x1)
```

This prints all node voltages and all currents except any current in the subcircuit x1.x1.

Output variables supported by Eldo are:

- Voltage at nodes, for example:

```
v(4) v(5,3)   v(2 4,5 out x1.1)
```

- Voltage on devices, for example:

```
E(Q1,BE,CE M1,DS)
E(C1,PN) E(C2)
```

  Note: reversing voltage is not supported yet in Eldo. i.e. E(Q1,EB) and E(C1,NP) will both be ignored.

  Note: default keyword handling is fully supported by Eldo. E(C1) is similar to E(C1,PN). But the voltage may not exist in Eldo, so E(M1) will be ignored in Eldo as it is similar to E(M1,DG).

  The supported keywords are:

Table 25-7. Supported Keywords for Voltage on a Device

Element type	Supported keywords
JFET	DS, GS
MOSFET	DS, GS, GB, BD, BS
BJT	CE, BE, BC

- Current in a device terminal, for example:

```
I(Q1,C,B,E D J1,D,G,S M1,D,G,S,B XA.Q1,B,C)
```

  Note: default keyword handling is fully supported by Eldo.

- Current density in a device

  Not supported for resistors, diodes, JFETS, MOS and BJTS, otherwise its is equivalent to current.

- Power, for example:

```
P(Q1 Q2)
```

  Note: Time averaging is not currently supported. P(Q1,TA) will be ignored.

- Noise variables, for example:

```
INOISE ONOISE(DB) ONOISE
```

When in TIspice compatibility mode the output variable names are assumed to be in the TIspice format. To keep the Eldo format with TIspice compatibility mode you must use the option **`TIELDOOF`**.

So in TIspice compatibility mode and with option **`TIELDOOF`** specified, the following statement will give an error:

```
.print dc E(D1,PN R1)
```

.PUNCH command

The TIspice **.PUNCH** command is synonymous with the Eldo **.PROBE** command.

.FOUR command extension

This calculates the fourier coefficients for the sinusoidal harmonic components of any output variable in the circuit. This command creates a table containing:

- DC component
- FOUR_NCOEFF first Fourier coefficients
- Total harmonic distortion THD

.FORCE command

Forces one or more nodes to specified voltage(s) with respect to ground for the initial transient solution. This is similar to the Eldo .IC command, values will be used only for the DC performed prior to TRAN analysis. Both commands are used to give values replacing the DC solution. With the **.FORCE** syntax, initial values of inductors current can be given while with Eldo **.IC** only node voltages can be initialized including nodes inside subcircuits. Syntax:

```
.FORCE  [NODE] node_name1 value1... node_name2 value2...
+                 [IND|OBJECT] object_name1 value1...
```

node_name	Node name.
VALUE	Value at node.
IND|OBJ	Inductor or object. For objects, only voltage source components can be specified to set the current on.
object_name	Inductor or voltage source component.

.INITIAL command

Specifies estimated solutions of DC analyses to improve DC convergence. This is similar to the Eldo .NODESET command, but analysis type can be specified, and **.INITIAL** can also be applied to impose **.NODESET** conditions on devices. Syntax:

```
.INITIAL ANALYSIS [NODE] node_name1 value1... node_name2 value2...
+                 [IND|OBJECT] object_name1 value1...
```

ANALYSIS	Type of analysis can be OP, TR or DC[SWEEP].

OP
Initial conditions for operating point analysis.

TR
Initial conditions for transient analysis.

DC
Initial conditions for DC analysis.

node_name	Node name.
VALUE	Value at node.
IND|OBJ	Inductor or object. For objects, only voltage source components can be specified to set the current on.
object_name	Inductor or voltage source component.

Examples:

```
.INITIAL OP A 5V
.INITIAL TRANSIENT A 4V
```

When an OP analysis is being performed, 5V will be used as a NODESET value. When a TRAN analysis is being performed, 4V will be used as a NODESET value

```
.INITIAL DC IND L1 5m
```

During a DCSWEEP analysis, a value of 5m will be used as NODESET value for the current flowing through object L1.

.ECHO command

Echo on/off is used to switch on/off the netlist lines to write to the output (*.chi*) file. The netlist lines between the echo off/on statements are disabled.

```
.ECHO ON|OFF [<string>]
```

The optional string on the echo command is always dumped.

Example:

```
*test ti
.param n=1
.param k=0
.param k1=n**k
.echo off "END OF DUMP"
v1 1 0 #n**k#
R1 1 0 1k
.echo on "RESTART THE DUMP"
.op
.print op  v(1)
.end
```

The part of the netlist between echo off and echo on will not be written in the output file because echo has been disabled (**`.echo off`**). The last part will be written because echo has been re-enabled (**`.echo on`**).

Netlist

Model selection using parameter string

```
X1 ... MOD = foo
M1... %MOD% ...    -> model foo will be used
```

Value "x"

x indicates unit 1e6 i.e. 2x represents 2.0e6. MEG is required in Eldo.

Expressions

Mathematical expressions can be enclosed in pound signs (#) or single-quotes (').

Parameter statements

Two ways of writing TIspice parameter statement allowed:

```
P pname [=] value [TC=<TC1> [,<TC2>]]
```

or:

```
.PARAM pname value [TC=<TC1> [,<TC2>]]
```

PR, PL, and PC are pre-defined parameter names that act as scaling factors for all resistors, inductors, and capacitors respectively in the circuit. These default to 1 and can be changed using this statement. These can be defined local to subcircuits and also passed as subcircuit parameters. Use extreme caution in using these parameter names, as they change the values of all R, L, and C elements in the circuit.

Functions and Sources

Delay

DELAY is equivalent to TD or SHIFT in PWL signals.

Sources

Current and voltage sources SIN, SFFM, AM, EXP allowed.

Dangling Nets Options

ALLOW_DANGLING_NETS causes Eldo to find all nets in the circuit with only one connection and connect each one to ground through a resistance of 1/GMIN.

NOALLOW_DANGLING_NETS causes Eldo to give a topology error and stop if any nets are found with only one connection.

Eldo Extensions for TIspice Compatibility

.DC command extensions

- Sweep of gain for linear controlled sources

  The **.DC** command has been extended to support the sweep of gain for linear controlled sources VCVS, CCVS, CCCS, VCCS. Only the linear case is supported and only the gain of these sources may be swept. Example:

  ```
  .dc E1 3 5 0.5
  ```

  The gain of the VCVS E1 will be swept from 3 to 5 in increments of 0.5.

  See ".DC" on page 588.

- Third level dc sweep

  The DC sweep can be nested up to three levels deep. Example:

  ```
  .dc v1 2 3 0.5 v2 5 9 0.5 v3 -1 2 0.1
  ```

.DEFAULT command

This command resets the default values for elements, device initial conditions and model parameters.

The general form for resistors, capacitors, and inductors is:

```
.DE[FAULT] TYPE VALUE
```

The general form for other types is:

```
.DE[FAULT] TYPE {KEYWORD [VALUE]}
```

Eldo implementation of **.DEFAULT** does not support the Lossy Transmission Line (LDTL) model and IC for active devices are ignored.

See ".DEFAULT" on page 600.

.MEAS command extension

For the trigger and target specification formats of the **.MEAS** command, SIG_H and SIG_L are also available. These represent the High and Low signal values respectively. Default high and low values are trig_val for the trigger signal and targ_val for the target signal. These high and low values are used to validate a transition before incrementing the cross, rise or fall counter.

These specifications are allowed in both standard Eldo mode and TIspice mode.

See ".MEAS" on page 717.

.MSELECT command

Automatic model selection. This command allows you to select model automatically for MOS devices. The selection is based on:

- the size and temperature of the specific device (W, L, TEMP)
- the size and temperature constraints of each model in the list provided (WMIN, WMAX, LMIN, LMAX, TEMPMIN, TEMPMAX)

Syntax:

```
.MSEL[LECT] dummy [models] mod1 [mod2 [mod3 [...]]]
```

It is not allowed to have a model statement with the same name as an mselect dummy model name.

.SCALE command

This command scales device and model parameters of active devices automatically.

The general form for devices (elements) is:

```
.SC[ALE] ELTYPE KEYWORD VALUE [KEYWORD VALUE ...]
+ [ELEMENTS ALL|EXCEPT] [ELNAME1 ELNAME2 ...] [(ELNAME1 ELNAME2)]]
.SC[ALE] ELTYPE KEYWORD VALUE [KEYWORD VALUE ...]
+ MODELS MODNAME1 [MODNAME2 ...]
```

The general form for devices models (models) is:

```
.SC[ALE] MODTYPE KEYWORD VALUE [KEYWORD VALUE ....]
+ [MODELS ALL|EXCEPT] [MODNAME1 MODNAME2] [(MODNAME1 MODNAME2...)]]
```

An additional feature is parameter scaling:

```
.SC[ALE] P FACTOR=VALUE [SUBCKT=SUBNAME] [INST=INSTNAME]
+ [PARAMS ALL|EXCEPT] [PARAM1 PARAM2 ...] [(PARAM11 PARAM22)]]
```

The element parameters or devices are only scaled at the parsing level.

If a **.DC** is used on a device element with a scale factor, the device element takes only the value specified by the DC analysis. There is no scaling effect.

The naming convention of devices, parameters, and models follows the Eldo naming convention if the **TISPICE** option is not set. The TIspice naming convention is followed if the option is set. TIspice uses partial names instead of wildcards, i.e. MOD1 means all names beginning with MOD1: MOD12, MOD1TT, MOD1, MOD1_A11, for example. In Eldo mode wildcards must be specified: MOD1*.

EBIT and PBIT source functions

The TIspice EBIT and PBIT source functions have been implemented. These represent exponential pulse with bit pattern and trapezoidal pulse with bit pattern sources respectively.

The full TIspice specifications are used in TIspice mode. However, standard Eldo also has access to this command, with the following changes:

- the TIspice optional parameter TD is specified as a mandatory first argument in Eldo
- the TIspice optional parameter NONPERIODIC is replaced in Eldo by the R keyword which can be found also on the original PATTERN Eldo statement
- the $ sign which must be set before the pattern string in TIspice is not required in Eldo
- Eldo does not allow specifying a file containing the bit pattern

For example, in TIspice, the statement:

```
v1 1 0 ebit 1 5 1n 0.5n 0n 1n 5n $101011101000011
```

is written in Eldo standard mode as:

```
v1 1 0 ebit 1 5 0n 1n 0.5n 0n 1n 5n 101011101000011 R
```

See “Exponential Pulse With Bit Pattern Function” on page 348.

Spectre Compatibility

Eldo can use a Spectre netlist as input. There are two flows to simulate a Spectre netlist in Eldo:

- Spectre compatibility flow

  Run Eldo on the Spectre netlist. Eldo will call the *spect2el* script to convert netlists from Spectre format (Spectre language syntax) into Eldo format (Eldo syntax). Additionally, *spect2el* generates several files that will be used to map the Spectre name to a SPICE name, and to avoid the reconversion of the Spectre files if nothing has changed in them. This flow is more straightforward and more efficient. This is the preferred approach.

- Spectre to Eldo Converter flow

  This flow is based on the *spect2el* script that converts libraries and netlists from Spectre format (Spectre language syntax) into Eldo format (Eldo syntax). You then run Eldo on the converted netlist. This flow offers more flexibility, in particular if manual modifications are necessary to workaround unexpected conversion issues. See Spectre to Eldo Converter for more information on the *spect2el* script.

Usage

The Spectre netlist can be specified on the Eldo command line with the following syntax:

```
eldo -sp spectre_file [-s2emode 2|1] [-i spice_command_file]
[-clean_sp] [-spectre_out pathname] [-sp_plot 2|1|0]
```

- **-sp** spectre_file

  The filename of the Spectre format netlist to be used as input for Eldo. *spectre_file* must be specified with an extension. No guess is made on the possible extension. If the filename is specified without its extension it will lead to an error.

- **-i** spice_command_file

  The filename of the Eldo/SPICE command file to be used in association with the Spectre netlist. Optional. Used with the **-sp** argument when a Spectre netlist is the input. *spice_command_file* must be specified with an extension. No guess is made on the possible extension. If the filename is specified without its extension an error will be generated. The *spice_command_file* must not include the *spectre_file*. Eldo will automatically perform the link between these files. Including the Spectre file will not work and the simulation will stop.

- **-s2emode 2|1**

  Used to switch between the old (pre-2009.1) and new converter. Optional. Used with the **-sp** argument when a Spectre netlist is the input. Set to 1 to run the old converter. Set to 2 (default) to run the newer one, which provides speed and robustness advantages. If errors occur when **-s2emode 2** is specified, a file named *spect2el.error* will be generated clearly describing the errors.

- **-clean_sp**

  Removes all the files generated by *spect2el*. As a consequence, if you want to relaunch the simulation on that design, the converter will reproduce the work. This will increase the overall simulation time. Optional. Used with the **-sp** argument when a Spectre netlist is the input.

- **-spectre_out** pathname

  Define the path in which the files generated by *spect2el* reside. Optional. Used with the **-sp** argument when a Spectre netlist is the input. Eldo creates *pathname* automatically if it does not exist. If **–clean_sp** is also specified, Eldo clears the files/subdirectories under *pathname*, while *pathname* itself is not removed after simulation. By default, *pathname* is identical to the one specified by **–outpath**.

- **-sp_plot 2|1|0**

  Controls the conversion of Spectre `save` statements into Eldo **`.PRINT`**, **`.PLOT`** and **`.PROBE`** commands as below:

  if `-plot 0`, the converter will convert the `save` statement into **`.PRINT`** (default).
  if `-plot 1`, the converter will convert the `save` statement into **`.PLOT`**.
  if `-plot 2`, the converter will convert the `save` statement into **`.PROBE`**.

Note

Eldo launches *spect2el* with the following options:
`-do_va 1 -netlist_converter 1`
You cannot change these options when running from the Eldo command line.

Description

spectre_file and *spice_command_file* must be specified with their extensions. No attempt is made by Eldo to surmise the possible extension. If the filename is specified without its extension an error will be generated. See Table 25-9 for example error messages.

The convention for the generated files is as follows:

- the converted SPICE files are written to the folder named *[outpath].spectre_file_dir/*
- the files generated for name mapping purpose are written to the folder named *[outpath].spectre_file_map_dir/*
- the files generated to avoid the (re)conversion of the Spectre files, when it is not necessary, are written to the folder named *[outpath].spectre_file_db_dir/*

You can manually remove these folders if they are no longer required. An alternative output directory can be specified with the **-spectre_out** *pathname* flag.

The *spice_command_file* must not include the *spectre_file*. Eldo automatically performs the link between these files. Attempting to include the Spectre file will not function, and the simulation will stop. See Troubleshooting Include Files for examples.

All SPICE commands (**.PLOT**/**.PROBE**/**.STEP**/**.EXTRACT**, and so on) can be used except for the **.ALTER** command, where the mapping is not handled. When specifying a name for the **.PLOT** command, the Spectre names can be used.

Note

Be aware that every device name or node name is case insensitive because SPICE language is being used. This differs from Spectre which is case sensitive.

For example, if the Spectre design contains:

```
aRes n1 n2 resistor r=1k
ares n1 n2 resistor r=2k
```

Eldo (SPICE) is case insensitive, so it will consider resistor “ares” has been defined twice. Spectre is case sensitive so differentiates between “aRes” and “ares.”

Here follows a simple example:

spectre_file.sp:

```
simulator lang=spectre

subckt foo a b
Aresfoo a b resistor r=1k
ends

subckt bar a b
Aresbar a b resistor r=1k
myInstance a b foo
ends

anotherInstance 1 0 bar
```

spice_command_file.cir:

```
*first line
c1 1 0 1p
I1 1 0 1
.plot tran I(anotherInstance.*)
.plot tran v(c1)
.tran 1n 10n
.end
```

Eldo can be launched on the Spectre design by entering the following:

```
eldo -sp spectre_file.sp -i spice_command_file.cir
```

In the *.wdb* output file, the name of the plotted elements are as defined in the Spectre design. In this example, I(ANOTHERINSTANCE.ARESBAR) will be plotted.

Spectre Default Constants and Functions

When using Eldo with the command line flag **-sp**, or if option **`USE_SPECTRE_CONSTANT`** is specified in the SPICE netlist, Eldo understands a number of Spectre default parameters, constants, and functions as follows:

- Spectre instance parameter `trise` is handled as an alias to `dtemp` for all instances
- Spectre instance parameter `perim` is handled as an alias to `peri` for diode instances
- Spectre default constants and functions as listed in Table 25-8 are allowed:

Table 25-8. Spectre Constants and Functions

Constant/Function	Value
m_e	2.7182818284590452354
m_log2e	1.4426950408889634074
m_log10e	0.43429448190325182765
m_ln2	0.69314718055994530942
m_ln10	2.30258509299404568402
m_pi	3.14159265358979323846
m_two_pi	6.28318530717958647652
m_pi_2	1.57079632679489661923
m_pi_4	0.78539816339744830962
m_1_pi	0.31830988618379067154
m_2_pi	0.63661977236758134308
m_2_sqrtpi	1.12837916709551257390
m_sqrt2	1.41421356237309504880
m_sqrt1_2	0.70710678118654752440
m_degperrad	57.2957795130823208772
p_q	1.6021918×10^{-19}
p_c	2.997924562×10^{8}
p_k	1.3806226×10^{-23}
p_h	6.6260755×10^{-34}
p_eps0	$8.85418792394420013968\times10^{-12}$

Table 25-8. Spectre Constants and Functions

Constant/Function	Value
p_u0	m_pi×4.0×10^{-7}
p_celsius0	273.15
f_mod(a,b)	{a-b×int((a+0.5)/b)}
atan2(a,b)	{atan(a/b)}
hypot(a,b)	{sqrt(a×a + b×b)}

These constants can be used inside all expressions.

Limitations

- The `.ALTER` command is not supported.
- When Eldo is using a Spectre netlist as input, and the netlist contains an internal node and a device with the same name, it is impossible for Eldo to distinguish between these. When Eldo analyses a subcircuit in the Spectre netlist, each element is attempted to be converted, without knowing if it is a node name or a device name. This can lead to plots not being performed as expected, because Eldo will not know what kind of plot is required (I or V for example). See Troubleshooting Spectre Node and Device Names for examples.

Troubleshooting

Include Files

The *spice_command_file* must not include the *spectre_file*, and vice versa. Eldo automatically performs the link between these files. Attempting to include the Spectre file will not function, and the simulation will stop.

- the SPICE file includes the Spectre file

  Eldo will not be able to parse the Spectre file, leading to the following syntax error message (example):

  ```
  ERROR  208: In file "./t1.sp" line 5:
  +   OBJECT "AOP": Unrecognized character or word A
  ```

  In this example, Eldo considers that “Aresfoo” in line 5 is requesting an aop (opamp).

- the Spectre file includes the SPICE file

  If you have the following top Spectre file, where *t1.cir* should be the *spice_command_file*:

  ```
  simulator lang=spectre
  ```

```
subckt foo a b
   Aresfoo a b resistor r=2k
ends

subckt bar a b
   Aresbar a b resistor r=1k
   myInstance a b foo
ends

anotherInstance 1 0 bar

simulator lang=spice
.include "t1.cir"
```

The Spectre to Eldo converter will not understand the **`.INCLUDE`** statement and so will not change its path nor copy the *t1.cir* file into the *.t1.sp_dir* folder. Eldo might find it, and proceed with the include, however it could also fail to find it and raise an error message.

Moreover if the file is both included and specified using the **-i** command line option, you will have duplicate definitions, as the file is effectively included twice.

Spectre does not understand SPICE syntax, which means only a full Spectre design can be used. However *spectre_file* could contain subcircuit definitions and Spectre library inclusion, and have its full design in the *spice_command_file* which may include some SPICE libraries.

To summarize, from one side a full Spectre tree is allowed and from the other a full SPICE tree. That can be a way to specify complex design mixing both SPICE and Spectre.

Spectre Node and Device Names

When Eldo is using a Spectre netlist as input, and the netlist contains an internal node and a device with the same name, it is impossible for Eldo to distinguish between these. When Eldo analyses a subcircuit in the Spectre netlist, each element is attempted to be converted, without knowing if it is a node name or a device name. This can lead to plots not being performed as expected, because Eldo will not know what kind of plot is required (I or V for example).

If *topsp*, the Spectre input file, contains a subcircuit instance named *x1*, and the SPICE command file, *cmd.cir*, contains:

```
.plot dc v(x1.c1)
```

then c1 must be an internal node name as a voltage plot is requested.

However, if *cmd.cir* contains:

```
.plot dc I(x1.c1)
```

then c1 must be a device name as a current plot is requested.

When Eldo is using a Spectre netlist as input, Eldo does not initially know the kind of plot requested (I or V). Therefore, when Eldo analyses *x1.c1*, each element is attempted to be converted; *x1.c1* will be mapped to *xx1.yy*, where *yy* can be *cc1* if there was a capacitor named *c1* in that subcircuit in the Spectre design, or *c1* if there was no capacitor named *c1*.

So if the following is specified:

```
.plot dc v(x1.c1)
```

it will be equivalent to .plot dc v(xx1.cc1) if c1 is defined as a device, no matter if it is a node name or not. In this case, the plot will not be performed.

Error Messages

Table 25-9. Spectre Conversion Error Messages

Error Code	Message	Description
5	Unable to open the temporary file "a_name".	Eldo cannot create a temporary file required to process the simulation. The file contains the converted Spectre design and the SPICE command file, if there is one.
33	Spectre file not specified.	Option -sp has been specified without a filename (eldo -sp).
1602	outpath not specified.	Option -outpath has been specified without a filename (eldo -sp topsp.sp -outpath).
1603	Spectre input file is a directory.	The specified Spectre file is in fact a directory name.
1604	Spectre input file "a_name" doesn't exist.	Eldo cannot find the specified Spectre file. For example, if the file extension was not specified or if the file does not exist.
1605	Error: Unable to create the directory "a_name". Either you don't have write access or there is not enough space left on device.	*spect2el* cannot create the directory that will contain the converted design.
1606	preprocessing of the Spectre files failed. Please refer to <path>/spect2el.log file for more details.	*spect2el* did not convert the Spectre design properly. Some missed cases may remain (when using the alterblock in Spectre for example). As above but with file and path information displayed.

Table 25-9. Spectre Conversion Error Messages

Error Code	Message	Description
1606	preprocessing of the Spectre files failed. Please refer to spect2el.log file for more details.	As above but when the Eldo -outpath argument is not specified.

Spectre to Eldo Converter

The Spectre to Eldo converter is a tool script that converts libraries and netlists from Spectre format (Spectre language syntax) into Eldo format (Eldo syntax). The script is named *spect2el*. See "Usage" on page 1407.

Prerequisites

- Korn shell must be installed under the */bin* directory. The tool will not be able to run if */bin/ksh* cannot be invoked.
- The "gawk" utility. This is provided along with the tool.

Supported Features

The tool can convert Spectre syntax to guarantee compatible results for:

- Basic component instantiations
- Device models and instances
- Subcircuit definitions and instances
- nport instances
- Controlled sources
- Functions and "if" statements
- Statistics
- Conversion of some of the device models written with SPICE syntax (simulator lang=SPICE). These models are: MOSFET models (nmos and pmos) MOS1, MOS2, MOS3, BSIM1, BSIM2, BSIM3, BSIM3v3, and BSIM4 (Spectre levels 1, 2, 3, 4, 5, 10, 11, and 54 respectively); BJT model (npn, pnp, lnpn, and lpnp), Spectre BJT (equivalent to Eldo level 1 as Spectre does not support level for BJT); Diode model (Spectre levels 1 and 3).

The tool can handle libraries with nested includes for any number of levels and for any directory structure.

The following Spectre device models are supported:

- BSIM3v3 MOS model
- Philips PSP 102 MOS model

  The following models are supported: psp1010, psp1020, pspnqs1020, psp1011, psp1021, pspnqs1021, psp101, psp102e, and pspnqs102e
- Philips MOS Model 11 Level 1102

  The following models are supported by the converter as Eldo Level 69: mos 11021, mos11021t, mos11020, mos11020t, mos1102e, and mos1102et
- Philips MOS Model 11 Level 11010
- VBIC bipolar model
- HICUM bipolar model
- BJT level one model (Gummel Poon)
- Diode level one model
- JFET level one model
- MOS1 model
- Polynomial models for R, L, and C
- Geometric Resistor model
- Physical Resistor model (phyres)
- MOS0 model. (However, since this model has no equivalent in Eldo, it is mapped to the MOS level=1 model of Eldo. Full compatibility is not guaranteed.)
- Bsource instances (r, l, c, i, v, q, g, and phi sources).

Notes

- Other device models are converted from the syntax point of view only with primary compatibility testing. The list includes: BSIM4, DP500, MOS2, MOS3, BSIM1, BSIM2, BSIM3v2, EKV, BSIMSOI3 PD, GAAS, TOM2, BJT504, BJT504t, JFET level 2, JFET level 4, PSP, TFT Polysilicon, TFT Amorphous, Mextram504, HICUM Level0, MOS2002, JUNCAP, JUNCAP2, JUNCAP_ELDO.
- For proper conversion of netlists or libraries, all the included files should exist. If any instance in the input library or netlist refers to a model, a subcircuit, or a Verilog-A module, which is not defined in the input library/netlist or one of its included files, this instance cannot be recognized or converted and a warning message will be displayed in the running terminal. The conversion will continue if running the tool in standalone mode. However, if running the tool from inside Eldo, the converter will exit with error

code 10 informing that this line is not converted and the conversion process will not continue until this error is corrected. In the new converter, `-s2emode 2`, a file named *spect2el.error* will be generated clearly describing the errors.

Netlist Conversion

The netlist conversion features come with the *spect2el* tool. This feature automates the netlists conversion process from Spectre syntax to Eldo syntax. The following features are supported:

Analyses

- AC analysis
- DC analysis
- Transient analysis
- S parameters analysis
- Monte Carlo analysis
- Sweep analysis
- Noise analysis
- Sensitivity analysis
- Transfer function analysis
- Stability analysis

Note

The above analyses types are partially supported. There are some combinations that are not yet supported.

Control statements

- Option statements:

Reltol	Vabstol	Iabstol	Temp
Tnom	Scalem	Scale	Gmin
Digits	Pivrel	Pivab	

- Paramset statement
- Save statement
- Assert statement

With the assert statement, you can set custom characterization checks to specify the safe operating conditions for your circuit. The Spectre circuit simulator then issues messages telling you when parameters move outside the safe operating area and, conversely, when the parameters return to the safe area.

The conversion of assert statements that use parameters inside expressions like below is not yet supported:

```
check assert expr=p2/p1 min=2 max=3
```

Using model types as a primitive with the assert statement like below is not yet supported:

```
check assert primitive=bsim3v3 param=l min=10u max=100u
```

- Simple combinations of alter and set statements

Sources

- Independent Sources:
    - Isource
    - Vsource
- Controlled Sources:
    - VCVS
    - VCCS
    - CCVS
    - CCCS
- Polynomial Controlled Sources:
    - Pcccs
    - Pccvs
    - Pvccs
    - Pvcvs

Components

- Primitive components (resistor, capacitor, diode, inductor, mutual inductor)
- MOSFET, JFET, Bipolar transistors
- Relay

The four-terminal relay is a voltage controlled relay tied between terminals t1 and t2. The voltage between terminals *ps* and *ns* controls the relay resistance. The relay resistance varies nonlinearly between *ropen* and *rclosed*, the open relay resistance and closed relay resistance, respectively. These resistance values correspond to control voltages of *vt1* and *vt2* respectively.

- Ideal switch

  Single-pole multiple-throw switch with infinite *off* resistance and zero *on* resistance. The switch is provided to allow you to reconfigure your circuit between analyses.

  When the switch is set to position 0, it is open. In other words, no terminal is connected to any other. When the switch is set to position 1, *terminal 1* is connected to *terminal 0*, and all others are unconnected. When the position is set to position 2, *terminal 2* is connected to *terminal 0*, and so on.

  The switch can change its position based on which analysis type is being performed using the xxx_position parameters.

- Transformer
- Delay Line

  The delay line model is a four-terminal device with zero output impedance and infinite input impedance. The output between nodes *p* and *n* is the input voltage between nodes *ps* and *ns*, delayed by the time delay *td* and scaled by gain.

- A2D

  The analog-to-logic converter transfers analog waveforms to a logic simulator.

- D2A

  The logic-to-analog converter converts a binary signal from a logic simulator to an analog waveform.

- Verilog-A instances
- Current probe (iprobe)
- Independent resistive source (port)

Limitations

- The following features cannot be converted with the tool. They require some manual modifications.
  - o Libraries and netlists must be case insensitive. Case sensitive libraries and netlists cannot be handled.
  - o Libraries and netlists must use Spectre’s syntax (simulator lang=spectre). Any SPICE syntax should be preceded by “`simulator lang=spice`” to be correctly

handled. To include Eldo library files, precede the syntax by "`simulator lang=eldo`" to be correctly handled.

- Nested sweeps that sweeps more than one paramset. For example:

```
sweep1 sweep paramset=paramset1 {
   sweep2 sweep paramset=paramset2 {
      ....
   }
   }
```

- Complicated combinations of the alter set statements.
- Statistical data defined in a separate file with the standard deviation set to a parameter value with this parameter defined in the same separate file will cause simulation errors when the parameter affected by this statistical data is in another file. The problem can be avoided by including this separate file in the netlist after conversion. This limitation is not existing now when using the new converter, `-s2emode 2`.
- In the PWL definition, a maximum of 500,000 pairs are accepted. If this number is exceeded, the conversion will fail.
- Long lines, it is advisable that any line should not be longer than 2,000 characters, usage of line continuation is strongly advised in the original netlist.
- Immediate set options and deferred set options are partially covered.

Usage

The syntax for the Spectre to Eldo Converter tool is as follows:

```
spect2el
-in_dir <dir1>
[-file_list <list>]
[-out_dir <dir2>]
[-log <file>]
[-remove_param 1|0]
[-remove_param_warn 1|0]
[-do_va 1|0]
[-inline_warnings 1|0]
[-netlist_converter 1|0]
[-devx 1|0]
[-scs2lib 1|0]
[-condition 2|1|0]
[-database_dir <db_dir>]
[-plot 2|1|0]
[-s2emode 2|1]
[-css 1|0]
```

All the above arguments are optional except for `-in_dir`.

`-in_dir dir1`	Input directory where the library/netlist to be converted reside. This could be a full path or a relative path, relative to where the tool is initiated. **Required**.
`-file_list list`	File list to be converted. It defaults to all files in the given directory if not specified. Default ‘*’.

Note

The file list should only include the library main file(s), which is to be directly included in the netlist in case of library conversion, or only the netlist file in case of netlist conversion. Other files, which are included from within those main file(s) are handled automatically. In other words, each file in the file list should not be included in any other file in the file list.

Note

Each file in the file list should be a standalone file. It should not depend on any other file in the file list. That is, it should not use any model, parameter, subcircuit, which is not defined in it or in any of the files it includes.

`-out_dir dir2`	Output directory where the converted files are to be written. This could be a directory name or a path. This directory will be created by the tool. If this directory already exists a warning message is issued. If not specified, it defaults to the same name as the input directory name but with a suffix “*_converted*.” Default *./{in_dir}_converted* for output name.
`-log file`	Specifies the log file that contains a brief report about the conversion process including any error or warning messages appearing during the conversion. If not specified, it defaults to a file named *spect2el.log* in the current working directory. The file can be specified by name, or by full path, or relative path to the current working directory.
`-remove_param 1|0`	If set, it will remove some model card parameters which are not supported by Eldo. Default 1 (set). If set, parameters removed are: “dope” for Diode level 1. “meto” and “wnoi” for BSIM3v3.
`-remove_param_warn 1|0`	If set, a warning will be issued each time one of the above parameters is removed. (This works only if `-remove_param` is set.) Default 0 (not set).
`-do_va 1|0`	Attempts to convert Verilog-A instances used inside SPICE netlists. (Note, this feature is not 100% guaranteed. Its success is dependent on the syntax of the Verilog files included. Any unusual syntax in the Verilog file may cause

	this feature to fail. Any issue regarding Verilog syntax is beyond the scope of this converter.) Default 1 (set).
`-inline_warnings 1|0`	Useful converter warnings will be printed inline as comments in the converted netlist to ease any required manual conversion work. Default 0 (not set).
`-netlist_converter 1|0`	Enable the netlist conversion feature. Default 1 (set).
`-devx 1|0`	Control whether the mismatch variation (for Monte-Carlo analysis) in Spectre would be converted to *devx* variation (set to 1) or *dev* variation (set to 0) in Eldo. Default is 1 which means that it would be converted to *devx*.
`-scs2lib 1|0`	Enable the feature of converting all files with extensions *.scs* to have the extension *.lib*. This also handles all include statements. Default 0 (not set). To switch it on, set this option to 1.
`-condition 2|1|0`	Controls the conversion of Spectre statements that include the conditional operator "?" as below: if `-condition 0`, the converter will parse the conditional statement as is (default). if `-condition 1`, the converter will use keyword **eval** in the conversion (pre-2008.2 behavior). if `-condition 2`, the converter will use keyword **valif** in the conversion.
`-database_dir db_dir`	Allows you to make use of a library that has been converted before and is included in your netlist instead of having to convert it again for each design and for each time you modify your design. This is achieved by saving a database for the library when it is first converted, and then reading this database later to convert the netlists using this library. This database is written to or read from the path specified by *DB_dir*. You should have write access to this path. The converter will check for every file if it has an entry in this database or not. If a file has an entry it will not be converted but will be retrieved from the database instead. Otherwise, if the file does not have an entry in the database (or is more recent than the entry), this file will be converted and saved to the database. Note that *DB_dir* should be independent from the *out_dir*. *DB_dir* cannot be set to the same directory as *out_dir* or any of its subdirectories.
`-plot 2|1|0`	Controls the conversion of Spectre `save` statements into Eldo **`.PRINT`**, **`.PLOT`** and **`.PROBE`** commands as below:

	if `-plot 0`, the converter will convert the `save` statement into **`.PRINT`** (default). if `-plot 1`, the converter will convert the `save` statement into **`.PLOT`**. if `-plot 2`, the converter will convert the `save` statement into **`.PROBE`**.
`-s2emode 2|1`	Used to switch between the old (pre-2009.1) and new converter. Set to 1 to run the old converter. Set to 2 (default) to run the newer one, which provides speed and robustness advantages. If errors occur when `-s2emode 2` is specified, a file named *spect2el.error* will be generated clearly describing the errors.
`-css 1|0`	Used to choose either to convert all sections of your library or just convert the selected sections. Set to 0 (default) convert all sections. Set to 1 to convert only selected sections. (Note: setting to 1 might cause speed loss in some cases.)
`-h`	Display the tool usage.

Usage Examples

The following is an example of converting a design including the library:

```
spect2el -in_dir . -out_dir ./out -file_list netlist.scs -plot
```

The following is an example of converting a library:

```
spect2el -in_dir . -out_dir ./out -file_list allModels.scs -scs2lib 1
```

In both cases, the output directory *out* will be created if it does not already exist. A directory named *external files* will be created in the directory *out* for converted **`.LIB`**/**`.INCLUDE`** files.

Appendix A
Error Messages

Error Message Classification

When an error is detected during parsing of the simulation input file, a message is printed out specifying:

- The source line number.
- The text containing the error.

Warnings

Warnings may be caused by improper use of commands or parameters which are then ignored by the analyzer. A warning does not prevent the continuation of analysis and simulation. By default, Eldo displays each type of node connection fault warnings (107, 108, 113 and 252) only three times; to override this default use the **MSGNODE** option.

For more information on this option, please refer to "MSGNODE=VAL" on page 996.

Syntax Errors

Error messages usually result from commands or parameters which are not recognized by the analyzer. When a syntax error is detected the simulation does not continue. It is then necessary to edit <*circuit*>*.cir* and correct the syntax error. In the printout <*circuit*>*.chi*, the error location in the text is indicated by a "^" character, possibly accompanied by a message. The source line number and the source line may not be indicated if the syntax error is detected on a global basis without reference to a particular line.

Effects

In the event of a warning, execution continues. If a syntax error is detected on the first analysis step (lexical and syntax analysis), circuit parsing continues until the end of this step, otherwise execution is aborted immediately. The numbers of errors and warnings found are displayed at the end of the first and second step.

> **Note**
> After an error has been detected by the analyzer, subsequent error messages may be unfounded. It is therefore recommended to modify the source file to eliminate the first error before attempting to interpret the other messages.

Error Messages

Global Errors

Table A-1. Global Errors

Error Number	Description
Error 1	Unable to open the file *name*
Error 2	Unable to open the library file *name*
Error 5	Unable to open the temporary file *name*
Error 6	Nested #com statements are not allowed
Error 7	Unable to delete the temporary file *name*
Error 9	Internal error in AC analysis
Error 10	Non invertible matrix
Error 11	Bad execution of *name*
Error 12	Check sum incorrect. Check your key
Error 13	Error in reading key: (*number*). Check your key
Error 14	Error in reading ADC/DAC; file *name* not specified
Error 15	The environment variable USER is not set by the system. Set this variable in your .cshrc or .login script
Error 16	Unable to allocate *number* bytes. Memory already allocated *name*
Error 17	MEMALLOC: Unable to allocate *number* bytes. Memory already allocated *name*
Error 18	MAMALLOC: Unable to allocate *number* bytes. Memory already allocated *name*
Error 19	Unable to load file *name*. Analysis stopped
Error 20	The number of plots of the current simulation does not match the number of plots of the previous one. File reading stopped
Error 21	Storage capacity exceeded. Memory already allocated *name*, eldo_mem_used()
Error 22	Circuit nesting error

Table A-1. Global Errors

Error Number	Description
Error 23	Unrecognized character or word
Error 24	Unexpected end of file
Error 25	Line not consistent with language syntax
Error 26	No analysis specified
Error 27	Inductor/Voltage source loop found To allow a voltage loop made up of 0 voltage sources use .OPTION LOOPV0 To downgrade this error to a warning use .OPTION VOLTAGE_LOOP_SEVERITY = WARNING Voltage loops may lead to singular matrix during simulation
Error 28	Unable to reallocate *name*
Error 30	Mismatch in model specification for device *name*
Error 31	DATA or SWEEP specification: *name*
Error 32	Switch option cannot be used on device *name*
Error 34	.OPTIMIZE MOD: no parameter specified
Error 36	.OPTION LVLTIM must be 1, 2, 3, or 4
Error 38	Compilation errors in file *name*
Error 39	Not enough memory to handle waves created through the DEFWAVE command
Error 40	FML.exe not found
Error 41	*name*: Real value expected
Error 42	*name*: Non-real expression expected
Error 43	Only DC followed by TRANSIENT analysis is allowed.
Error 44	FasC model *name* already defined
Error 45	.OPTION DVDT must be -1 or 0.
Error 46	No plot to display for *name* analysis: Simulation stopped.
Error 47	Error in updating the library. Refer to file *name*
Error 48	No voltage or current AC input source specified
Error 49	Unable to spawn *name*: There might not be enough memory
Error 50	.OPTION NEWTON and OSR are not compatible
Error 51	No plot matching analysis type: Analysis stopped.
Error 52	No node “0” found in the circuit

Table A-1. Global Errors

Error Number	Description
Error 54	Syntax error parsing *name*
Error 55	The following line is too long
Error 56	No key to run *name*
Error 57	Nested DC Sweeps are not allowed within SimPilot
Error 58	Error: Command *name* not allowed within SimPilot
Error 59	No value given for parameter *name*
Error 60	Real value expected for *name*: End of line found
Error 61	.OPTION IEM and OSR are not compatible
Error 62	.OPTION IEM and NEWTON are not compatible
Error 63	Unable to include file *name*
Error 64	.MODDUP cannot be used in conjunction with LOT|DEV
Error 65	.MODDUP cannot be used in conjunction with .MC
Error 66	.OPTION HMIN: Value must be strictly positive
Error 67	.OPTION LVLCNV must be 0, 2, or 3
Error 68	COMMAND .INCLUDE/.LIB cannot find *name*
Error 69	Cannot find HDLA models
Error 70	.OPTION *name* expects an integer value
Error 71	TUNING: Unexpected parameter *name*
Error 72	Probable syntax error in library *name*
Error 73	Fatal error in *name* model, please check output file for details. Eldo Kernel can’t find the ‘SBVAL.PAR’ file
Error 74	Cannot find #endcom statement
Error 75	ADVance MS command must be used in place of Eldo in order to use VHDL-AMS models
Error 76	Internal error: Mismatch in parameter *name*
Error 77	Line not allowed inside command file
Error 78	Unknown command *name*
Error 79	DEBUG command number *name* not found
Error 80	.RUN: Unknown argument *name*
Error 81	Too many files opened

Table A-1. Global Errors

Error Number	**Description**
Error 82	Parameter not known: *name*
Error 83	Unable to alter *name*
Error 84	Missing parameters
Error 85	Syntax error or Syntax error at or near *name*
Error 86	Error evaluating *name*
Error 87	Command cannot be issued at run time, or Command *name* cannot be issued at run time
Error 88	*name* cannot change .MODEL card for that kind of element
Error 89	*name* disabled because AC analysis performed within transient analysis. However, with keyword ALL added on the .MC card, simulation should run
Error 90	SST analysis cannot be performed: Device *name* is not supported
Error 91	SST analysis cannot be performed: Gudm model on *name* not supported
Error 92	SST analysis cannot be performed: MOS model on *name* should be a charge-controlled model
Error 93	SST analysis cannot be performed: Non-Quasi Static effects on *name* is not supported
Error 94	Unknown directive: *name*
Error 95	No matching keyword *name*
Error 96	Parameter *name* cannot be evaluated
Error 97	Mismatch between .iic and .cir files: Simulation stopped
Error 98	Having both .STEP and SWEEP on AC/TRAN/DC cards is not allowed
Error 99	Unable to find a .DATA statement named *name*
Error 100	This version of Eldo does not include *name*

Errors Related to Nodes

The following error messages are preceded by `NODE <name>`:

Table A-2. Errors Related to Nodes

Error Number	**Description**
Error 101	No source on this node

Table A-2. Errors Related to Nodes

Error Number	Description
Error 102	Multiple input signal applied
Error 103	The DIGITAL to ANALOG Voltage Source Converter conflicts with another Voltage Source already attached to this node
Error 104	Multiple DTOA on this node and at least one of them is a Voltage Source Converter. This is not allowed
Error 105	Inconsistencies in the High Voltage specifications of the different RC DTOA Converters connected to this node
Error 106	Inconsistencies in the Low Voltage specifications of the different RC DTOA Converters connected to this node
Error 107	Node *name* not known
Error 108	Iout plot specification ignored on top-level nodes
Error 109	This node is a floating gate
Error 111	Less than two connections
Error 112	No DC path to ground
Error 113	A2D/D2A cannot be applied on non-existing nodes
Error 114	Connectivity around that node makes Matrix singular
Error 115	Related device not found
Error 116	Only IN port seen, and no signal applied
Error 117	Missing A2D/D2A to connect to object

Errors Related to Objects

The following error messages are preceded by `OBJECT <name>`:

Table A-3. Errors Related to Objects

Error Number	Description
Error 201	Cannot be used in AC analysis, or Cannot be used with CAPTAB
Error 202	AC input has not been found
Error 203	Parameter sweep not available for this object
Error 204	VTH1 and VTH2 are inverted
Error 205	Not found in file: *name*

Table A-3. Errors Related to Objects

Error Number	Description
Error 206	VHI and VLO are equal
Error 207	VHI and VLO are inverted
Error 208	Unrecognized character or word: *name*
Error 209	Incorrect declaration of POLY names
Error 210	Keyword FREQ expected
Error 211	Parameters are missing
Error 212	Unknown signal type: *name*
Error 213	Incorrect geometrical dimension: *name*
Error 214	No value specified for object: *name*
Error 215	Macro already defined
Error 216	Too many controlling nodes (> 3)
Error 217	Model not yet defined: model_name
Error 218	Model not found: model_name
Error 219	Short circuit element
Error 220	Voltage specifications are missing
Error 221	Is a multidimensional element
Error 222	SIN: Frequency below zero
Error 223	SFFM: FC below zero
Error 224	SFFM: FS below zero
Error 225	EXP: TAU1 below zero
Error 226	EXP: TAU2 below zero
Error 227	Unknown signal applied
Error 228	Either TD or F must be specified
Error 229	Multidimensional object. This variable is not declared
Error 230	Output pin is already a voltage source
Error 231	Output pin is already connected to a comparator output
Error 232	Geometries are below zero: please check DW and DL
Error 233	Value becomes zero. Exited
Error 234	Syntax error in the expression

Table A-3. Errors Related to Objects

Error Number	Description
Error 235	Brackets are missing in the TABLE values
Error 236	Table input values must be properly ordered
Error 238	Bus specification ignored
Error 240	A time or a value specification is missing for this bus
Error 242	This bus is already defined
Error 244	A signal has been already applied on this bus
Error 245	More than one .CHECKBUS applied on this bus
Error 246	No model specified in this functional call: *name*
Error 248	Unable to plot the capacitances of this device. Only charges are available
Error 250	Unable to plot the charges of this device. Only capacitances are available
Error 251	Unexpected character found
Error 252	Parameters or pins are missing for this device
Error 253	Is not accessible
Error 254	Unknown parameter: *name*
Error 255	Unknown keyword: *name*
Error 256	The sign : is missing after keyword
Error 257	‘(‘ found when not expected
Error 258	Element not found: *name*
Error 259	The following element is not a current source: *name*
Error 260	The following element is not a voltage source: *name*
Error 261	Pin not found: *name*
Error 262	Number of pins: *number*
Error 263	Number of controlling nodes: *number*
Error 264	Number of controlling zero voltage sources: *number*
Error 265	Number of controlled voltage sources: *number*
Error 266	Number of controlled current sources: *number*
Error 267	Number of parameters: *number*
Error 268	Error when initializing SOLVE routines
Error 269	Error when initializing INTEGRAL routines

Table A-3. Errors Related to Objects

Error Number	Description
Error 270	Parameter not found
Error 271	Duplicate definition of parameter: *name*
Error 272	Parameter already assigned to an equivalent: *name*
Error 273	Parameter index not found: *name*
Error 274	Error in macro *name*
Error 275	VSTATUS already called with different pin order on: *name*
Error 276	Unable to apply the function: *name*
Error 277	Unable to find previous state: *name*
Error 278	Error in function
Error 279	No model assigned to this element
Error 280	Model attached to this device is also attached to another device of a different type
Error 281	Its model is also attached to the following component: *name*
Error 282	Access resistor becomes less than or equal to zero
Error 283	Error when computing R value. Division by 0
Error 284	Keyword PARAM expected instead of: *name*
Error 285	Equal sign ‘=’ expected instead of: *name*
Error 286	Unknown Base specification
Error 287	Cannot get the current through non-voltage source
Error 288	Unable to parse expression of
Error 289	Radiation source not found
Error 290	SOI back source not found
Error 291	Radiation source specification error
Error 292	SOI back source specification error
Error 293	The current through cannot be used as argument
Error 294	Inconsistency in voltage specification
Error 295	Unexpected digit found in the bus: *name*
Error 296	Unable to code
Error 297	Unknown pin number
Error 298	R value is missing

Table A-3. Errors Related to Objects

Error Number	Description
Error 299	Mismatch in POLY specification
Error 801	Parameterizable object not created
Error 802	This RC line refers to a model which is already attached to a standard resistance: *name*
Error 803	This resistance refers to a model which is already attached to an RC line: *name*
Error 804	Probably an RC line: please, specify level = 3 in the relevant .MODEL card
Error 805	Missing model specifications
Error 806	Delay cannot be 0.0
Error 807	.PLOTLOG: Specified object is not 0xx instance
Error 808	.PLOTLOG: Functional instance not defined
Error 809	BSIM1: Parameter K1 = K10 + K1L/Leff + K1W/Weff <= 0.0. Check your model card or the transistor dimensions.
Error 810	.PBOUND: syntax is .PBOUND <pname> (mini,maxi)
Error 812	Object not yet created
Error 813	According to parameters specified, a model must be assigned to this device
Error 814	This device refers to an inconsistent model
Error 815	Inconsistency in thresholds/voltages declarations
Error 816	The thresholds must be in the range [VLO + 1%, VHI - 1%]
Error 817	The BS value for that core must be greater than the BS value
Error 818	The HC value for that core must be greater than zero
Error 819	The temperature-adjusted BS value for that core must be greater than zero
Error 820	The temperature-adjusted BR value for that core must be greater than zero
Error 821	The temperature-adjusted HC value for that core must be greater than zero
Error 822	The temperature-adjusted BS value for that core must be greater than the temperature-adjusted BR.
Error 823	AREA must be greater than 0
Error 824	FAS syntax expected on this element: HDL-A syntax found
Error 825	HDL-A syntax expected on this element: FAS syntax found
Error 826	LENGTH and AREA must be strictly positive
Error 827	Mismatch in port specification

Table A-3. Errors Related to Objects

Error Number	Description
Error 828	Equal sign '=' is missing in parameter list
Error 829	Unexpected equal sign '=' in parameter list
Error 830	Several ports with that index
Error 831	No port index specified
Error 832	Missing R value
Error 833	device_name COUPLING: expecting keyword I or V: found *name*
Error 834	Syntax error in PATTERN command
Error 835	Object Error
Error 836	Mismatch in coupling specification
Error 838	Already defined
Error 839	No model assigned to that device (MODDUP)
Error 840	Short circuit on dipole device
Error 841	This element cannot be converted into its switch equivalent
Error 842	Odd number of pins
Error 843	Declaration of that object must be put at the very end of the netlist, since it is connected to hierarchical nodes, otherwise the node may not exist yet.
Error 844	Pin specification is missing
Error 855	TDEV should be specified on that device
Error 856	Syntax HDLA_NAME = ELDO_NAME expected
Error 857	Multiple SST specifications
Error 858	Wrong object inside thermal network: only R/V/C device type can be specified
Error 859	AREA/PERI and W/L cannot be given together
Error 860	Multiple types of input applied
Error 861	Not declared
Error 862	Unable to mix FPORT syntax with RPORT syntax. FPORT must be replaced by a voltage current source with R/C/L specifications.
Error 863	RPORT can be applied on voltage source only
Error 864	Missing IPORT specification
Error 865	AREA is negative or 0.0
Error 866	First pin of that element is undefined

Table A-3. Errors Related to Objects

Error Number	Description
Error 867	Second pin of that element is undefined
Error 868	.TRAN and .FOUR specification is not allowed on the same device
Error 869	Level not specified for geometric model
Error 870	Generic/Parameter field appears twice
Error 871	Only 'VALUE', 'TABLE' or 'AC' arguments expected
Error 872	Device not found
Error 873	Missing L or W specification to enable appropriate model selection
Error 874	No character string allowed in Param/Generic list
Error 875	DDT or IDT operator allowed on 'E' or 'G' sources only
Error 876	L and W must be specified before M(<param>)
Error 877	Character not allowed on pattern
Error 878	Value not specified
Error 879	PORTS specification appears more than once
Error 880	PINS specification appears more than once
Error 881	Only PINS specification allowed on VHDL-AMS instances
Error 882	GENERIC specification appears more than once
Error 883	Dimension must be 1 exclusively
Error 884	Second controlling node must be same as the second pin
Error 885	Second controlling node must be same as the first pin
Error 886	Parameters of denominator for S/Z transform are all zero.
Error 887	More than one value specified for resistor value
Error 888	TSAMPLE not specified for PATTERN .CHECKBUS
Error 889	Undeclared inductor
Error 890	Expects syntax PWL(1) N1 N2 MODEL value
Error 891	Syntax error in IBIS model call
Error 892	Unknown IBIS device
Error 893	Component *name* is missing inside IBIS model call
Error 894	filename is missing inside IBIS model call
Error 895	MODEL and PIN can’t be both specified

Table A-3. Errors Related to Objects

Error Number	Description
Error 896	Cannot replace source, discontinuity found
Error 897	Expects an even number of parameters
Error 898	PZ or FNS allowed on E elements only
Error 899	Neither MODEL nor PIN specified
Error 900	FNS denominator cannot be 0

Errors Related to Commands

The following error messages are preceded by `COMMAND`

Table A-4. Errors Related to Commands

Error Number	Description
Error 301	.MC: Monte Carlo analysis is not compatible with .ALTER
Error 302	.MC: Monte Carlo analysis is not compatible with .STEP
Error 303	.ALTER: This command is not compatible with .STEP
Error 304	.MC: Unable to reallocate .MODEL field
Error 305	.MC: Internal memory error
Error 307	.CHRSIM: A previous transient analysis is required for the circuit
Error 308	.CHRSIM: A previous DC sweep analysis is required for the circuit
Error 309	.CHRSIM: No output found in previous circuit to be connected to input
Error 310	.SENS: Unable to create sensitivity matrix
Error 311	.SENS: Circuit size too large (>*name*) for sensitivity analysis
Error 312	.MC: Type not found
Error 313	.OPTFOUR: Unknown output format
Error 314	.INIT: Time values are missing
Error 315	.IC: Time values are missing
Error 316	.USE *name*: Specification unknown
Error 317	.TRAN: Parameters are missing
Error 318	.USE *name*: Specification unknown
Error 319	.PLOT *name*: This kind of analysis is not supported

Table A-4. Errors Related to Commands

Error Number	Description
Error 320	.LOOP: Object type not allowed: *name*
Error 321	.PARAM: Attempt to divide by zero
Error 322	.DC: W or L must be written in place of *name*
Error 324	.OPTIMIZE: *Name* or element *name* not yet defined
Error 325	.DC: Too many values specified
Error 326	.RESTART: The file used to restart has the same name as the current circuit. Rename the previous .cou file
Error 327	.OPTIMIZE: *name* not yet created
Error 328	.OPTIMIZE: Model *name* not yet created
Error 329	.OPTIMIZE *name*: Incompatible operation
Error 330	.DC: Unknown object *name*
Error 331	.OPTIMIZE: AC output node *name* not yet defined
Error 332	.IC: Error on *name*
Error 333	.SOLVE: Error in expression
Error 334	.SOLVE: Element *name* not yet defined.
Error 335	.OPTIMIZE: Parameter *name* not known
Error 336	.PLOT: Specification *name* not allowed
Error 337	.DC: Sweep object not specified
Error 338	.OPTIMIZE: Use ACOUT=<pin_name> to select output
Error 339	.NOISETRAN: Error with parameter
Error 340	The Thermal Noise Level (*number*) is not compatible with values extracted from MOS model
Error 341	.NOISE: Element *name* not found
Error 342	.SOLVE: Element *name* not found
Error 344	.MFTA: Parameter *name* not found
Error 345	.MFTA: Missing argument
Error 346	.SAVE: File *name* is already specified in a previous .SAVE command
Error 348	.INIT cannot be applied on node *name*: Conflict of signals
Error 350	.WCASE: Unknown analysis specification *name*
Error 352	Expression *name* can accept only items V(node) or I(object)

Table A-4. Errors Related to Commands

Error Number	Description
Error 354	Expression *name* can accept only items V(node), I(object) or W(w)
Error 359	.PARAM: “=” is missing in expression: *name*
Error 360	.OPTIMIZE: Unexpected expression: *name*
Error 361	.EXTRACT: Unexpected expression: *name*
Error 362	.STEP PARAM: Parameter *name* not specified
Error 363	.DC: The parameter *name* is not specified or is not a primitive. Unable to perform this analysis
Error 364	.STEP: Parameter *name* not defined at the top level
Error 365	.DC PARAM: Parameter *name* not defined at the top level
Error 366	.ALTER: Appears as the title
Error 367	*name*: Unknown command
Error 368	.OPTIMIZE: sign = < > expected instead of *name*
Error 369	.DEFWAVE: Error *name* =
Error 370	.OPTIMIZE: *name* not allowed on parameter
Error 371	.PLOTLOG command: ‘->’ sign is missing
Error 372	.PLOT command: Found *name* while expecting a (or)
Error 373	.SETSOA: Opening bracket expected after *name*
Error 374	.SETSOA: Closing bracket expected after *name*
Error 375	.SETSOA: Error in the expression *name*
Error 376	.SETSOA: Expected .SETSOA EXPR <expression> = (MIN,MAX)
Error 377	.SETSOA: Expected a * or a value for *name*
Error 378	.SETSOA: Model *name* not yet created
Error 379	.WC cannot be used with command .MC
Error 380	.CONNECT: Node *name* not yet created. The X instance which creates the node must be placed before the .CONNECT card
Error 381	.SIGBUS: No parameters allowed (*name*)
Error 384	.DISTRIB *name*: Opening bracket expected
Error 385	.DISTRIB *name*: Closing bracket expected
Error 386	.DISTRIB *name* is already defined
Error 387	*name*: Specified terms must be properly ordered

Table A-4. Errors Related to Commands

Error Number	Description
Error 388	.DISTRIB *name*: <probability> must be in the range [0 to 1]
Error 389	Distribution *name* referenced but not defined
Error 390	Error in LOT/DEV specification for expression *name*
Error 391	.PARAM: Double specification for *name*
Error 392	.TRAN: No parameter allowed: (*name* found)
Error 393	.DISTRIB *name*: <deviation> must be within the range [-1 to 1]
Error 394	Unable to measure ISUB (*name*): multiple connections
Error 395	.OPTION *name* ignored
Error 396	.STEP: DEC, LIN or OCT expected: *name* found
Error 397	.STEP: No parameter allowed: *name* found
Error 398	.STEP: DEC/OCT values must be positive
Error 399	*name*: Negative X value found for DEC scale
Error 400	.LMIN or .WMIN: Parameters are missing
Error 401	.OPTNOISE: Real value expected after keyword *name*
Error 402	.OPTNOISE: Keywords D V TD TV expected: *name* found
Error 403	.OPTNOISE: SV: Value must be less than 1 but greater than 0: *name* found
Error 404	*name*: The sign ‘,’ is not allowed
Error 405	.SOLVE: Cannot accept functions such as YVAL, TPD, MIN, MAX and INTEG
Error 406	.EXTRACT *name*: Cannot find a .EXTRACT with this label
Error 407	.OPTFOUR: Unknown parameter: *name*
Error 408	.EXTRACT *name*: Cannot find a .FOUR with the label *name*
Error 409	.SNF: Unknown parameter *name*
Error 410	.SNF: Double input specification for *name* field
Error 411	Multiple .SNF cards not allowed
Error 412	.SNF card: *name* specification expected
Error 413	.FOUR card: Syntax is [LABEL=] <four_label> <wave>
Error 414	.EXTRACT *name*: RMS accepts transient waves or AC noise waves only
Error 415	.OPTWIND or .OPTPWL: Parameter *name* cannot be changed
Error 416	.OPTWIND or .OPTPWL: Missing time-value for *name*

Table A-4. Errors Related to Commands

Error Number	Description
Error 417	.OPTWIND or .OPTPWL: Decreasing time-value for *name*
Error 418	.AC: Syntax error card: *name*
Error 419	.OPTWIND or .OPTWPL: Missing value specification at or near *name*
Error 420	.OPTWIND or .OPTWPL: *name*
Error 421	.IFT: Unexpected argument *name*
Error 422	.OPTNOISE: NOISE and NONOISE cannot both be specified
Error 423	.OPTNOISE: ALL=OFF and NONOISE cannot both be specified
Error 424	.OPTNOISE: ALL=ON and NOISE cannot be both specified
Error 425	.SNF: Element *name* not found
Error 426	.SNF: Element *name* appears as both input and output
Error 427	.OPTFOUR: Parameter FS must be specified
Error 428	.PZ: This analysis might give unexpected results because several AC sources are found in the circuit
Error 429	.ALTER: Not allowed in Accusim netlist
Error 430	.NOISETRAN: Too many runs specified
Error 431	.OPTFOUR: Bad value for parameters TSTART, TSTOP, NBPT and FS
Error 432	.STEP PARAM ( ): Syntax error
Error 433	.STEP PARAM ( ): Only LIST allowed for multi step
Error 434	.STEP PARAM ( ): The number of values does not match the number of parameters
Error 435	.STEP: TEMP keyword cannot appear more than once
Error 436	.STEP: Unknown *name*
Error 437	.CHECKSOA: Unexpected parameter
Error 438	.SETSOA: Expecting D, E or M: *name* found
Error 439	.STEP: Missing increment parameter
Error 440	.STEP: Missing boundary specification
Error 441	.WCASE: Missing analysis specification
Error 442	.AC LIST: Frequencies are not properly ordered
Error 443	.DC: Only a real value is expected
Error 444	.DEFWAVE: Duplicate definition of *name*

Table A-4. Errors Related to Commands

Error Number	Description
Error 446	.OPTFOUR: Unexpected expression: *name*
Error 447	.DEFMAC: Recursive definition of *name*
Error 448	.OPTFOUR: The expression for *name* could not be evaluated
Error 449	.OPTFOUR: TSTART > TSTOP
Error 450	.STEP LIST: *name*
Error 451	.NOISETRAN: Cannot be used with .OPTION SAMPLE
Error 452	.MEAS: Unknown analysis type *name*
Error 453	.MEAS: Unknown function *name*
Error 454	.MEAS: Unexpected wave *name*
Error 455	.MEAS: Unexpected keyword: *name*
Error 456	.MEAS: Missing keyword: *name*
Error 457	.MEAS *name*: Cannot find a .MEAS of that name
Error 458	.MEAS *name*: LAST not accepted
Error 459	.DC: *name* unknown
Error 460	.STEP: Increment is 0.0
Error 461	.SETSOA: unexpected keyword *name*
Error 462	.EXTRACT: FILE=*name* must be placed before the expression
Error 463	.STEP: multiple declaration for *name*
Error 464	.MCDEV: syntax E(devname[,geo]) expected
Error 465	.TRAN: decreasing time not allowed: *name*
Error 466	.LOTGROUP: double specification for *name*
Error 467	.FFT: Syntax is .FFT <wave> <FFT parameters>
Error 468	.PLOTBUS: specification error on *name*
Error 469	.MODEL *name* HOOK: syntax is .MODEL <name> HOOK [<logical_lib_name>:]<entity_name>[(<architecture_name>)]
Error 470	.MC specification on *name*: dead-zone too large compared to ‘normal’ zone
Error 471	Directive error *name*
Error 472	.PLOT: SMITH chart available for AC only
Error 473	.SETBUS: node *name* does not exist

Table A-4. Errors Related to Commands

Error Number	Description
Error 474	.AC expecting DEC, LIN, OCT, LIST or DATA: *name* found
Error 475	.PART: Unknown flag: *name*
Error 476	.PART: Syntax error. Syntax is .PART <flag> SUBCKT|INST (<list>)
Error 477	.SETBUS: node *name* does not exist
Error 478	.EXTRACT: such SST output .H extensions
Error 479	.HOOK: expects 'MOD=' statement
Error 480	.LOOP: unexpected string *name*
Error 481	.DEFWAVE *name*: expects .H extensions
Error 482	*name*: expects an integer value
Error 483	*name*: missing a '(' bracket
Error 485	.STEP: step windows must be separated
Error 486	Two ports are required for getting RNEQ/NFMIN_MAG/GOPT/ GAMMA_OPT/GAMMA_OPT_MAG/PHI_OPT/NC
Error 487	.SNF: error specifying SIDEBAND: *name*
Error 488	.PARAMOPT *name* 3 or 4 values expected
Error 489	.PLOT: (SMITH) or (SPECTRAL) incompatible with (VERSUS)
Error 490	.PLOT: (VERSUS) must be followed by only one wave.
Error 491	.PLOT: No wave found before (VERSUS)
Error 492	.TVINCLUDE: syntax error in test vector *name*
Error 493	.DATA: *name*: Unable to open file *name*
Error 494	.EXTRACT: *name*: LABEL = <> expected
Error 495	.EXTRACT: *name*: expects MEAS(label_name)
Error 496	.SNF: *name*
Error 497	*name* not allowed with Mach
Error 498	.PRINT/PLOT/PROBE/EXTRACT: bad number of tones specified
Error 499	.CORREL: parameter *name* cannot be found
Error 500	.CORREL: Instance *name* cannot be found

Errors Related to Models

The following error messages are preceded by `MODEL`:

Table A-5. Errors Related to Models

Error Number	Description
Error 501	Not found in library
Error 502	Unknown in the libraries
Error 503	Undeclared model reference
Error 504	Is defined twice
Error 505	Parameter unknown
Error 506	Model not yet defined
Error 507	Inconsistency in level specification. LEVEL: *number*
Error 508	Level not implemented
Error 509	R model not declared
Error 510	NSUB < NI. exited
Error 511	C model not declared
Error 512	G model not declared
Error 514	BIDIR type not allowed. Use ATOD or DTOA
Error 515	Default BIDIR Converter not found
Error 516	LEVEL already specified
Error 517	Parameter TOX must be greater than 0
Error 518	Parameter STRUCT must be +1 or -1
Error 519	Unacceptable parameter value
Error 520	MJSW must be less than 1.0
Error 521	N must be positive
Error 522	FC must be less than 1.0
Error 523	The following parameter must only be used when eldomos is active or level 12 or 13 are used
Error 524	Model Error
Error 526	Unable to alter the parameters of that model
Error 527	This model is used by Eldo-XL and cannot be replaced

Table A-5. Errors Related to Models

Error Number	Description
Error 528	This model cannot be changed using .LIB in an interactive mode: Inconsistent level or architecture
Error 529	The following parameter cannot be used with model MM9
Error 530	Unknown language type
Error 531	Double MC specification (via .MCMOD/.MODEL) for the parameter *name*
Error 532	Expects PWL(1) SYMMETRY NO SOURCE CARDS DATA
Error 533	Parameter COX cannot be specified for that model
Error 534	W model: missing declaration of parameter N
Error 535	W model: incorrect number of parameters
Error 536	parameter *name* already assigned
Error 537	vector size not consistent for parameter *name*
Error 538	Missing inout values for parameter *name*
Error 539	Size of vectors not consistent
Error 540	Missing call to MP_vect_array
Error 541	inconsistent dimension of the vector array
Error 542	Position already allocated
Error 543	PASSFAIL/BISECTION expected after METHOD parameter
Error 544	DEVX specification cannot be applied on .MODEL parameters
Error 545	This level is not supported yet

Errors Related to AMODELS

The following error messages are preceded by `AMODEL <name>`:

Table A-6. Errors Related to AMODELS

Error Number	Description
Error 601	Attempt to redefine an Amodel while the previous definition is not completed
Error 602	Definition not completed
Error 603	Macromodel not found
Error 604	Pins must be specified before Ports

Errors Related to Subcircuits

The following error messages are preceded by `SUBCKT <name>`:

Table A-7. Errors Related to Subcircuits

Error Number	Description
Error 701	Not found in library
Error 702	Undeclared subcircuit reference
Error 703	is defined twice
Error 704	Cannot be declared before the previous one has finished
Error 705	Incorrect number of nodes in call at line
Error 708	Recursive .SUBCKT calls are not allowed
Error 709	Unable to connect the output to a power supply or another digital output
Error 710	Error in declaration
Error 711	Already obtained from another LIB file
Error 712	Cannot define a parameter named ‘M’
Error 713	An ‘IF’ statement is not properly closed
Error 714	Unexpected keyword *name*
Error 715	refering to non-existing hierarchical node

Miscellaneous Errors

Table A-8. Miscellaneous Errors

Error Number	Description
Error 901	*number*: Unknown signal type at line
Error 902	*name*: Unknown parameter
Error 903	*name*: Specification ignored
Error 904	*name*: Unknown model type
Error 905	*name*: Parameter not found
Error 906	*name*: Unknown output format specified
Error 907	*name*: Can not affect value
Error 908	*name*: Parameter not yet set

Table A-8. Miscellaneous Errors

Error Number	Description
Error 909	*name*: Unknown specification
Error 910	*name*: Model not yet defined
Error 911	Diagram not supported at line
Error 912	*name*: Unknown command or component at line
Error 913	Decreasing time on input signal number *name*
Error 914	Decreasing time on input signal assigned to object
Error 915	CFAS: Call to get_param_value() allowed in ALLOCATE MODE only
Error 916	CFAS: Call to get_param_value_b4() not allowed
Error 917	Unable to read from file *name*
Error 918	Cannot reallocate component table
Error 919	Cannot reallocate input signal
Error 920	Unable to parse expression *name*
Error 921	*name* is not specified in ATOD model
Error 922	Description of DTOA *name* is missing
Error 923	DTOA *name* is declared as ATOD
Error 924	Description of ATOD *name* is missing
Error 925	ATOD *name* is declared as DTOA
Error 926	Error in expression *name*. I or V expected
Error 928	BIDIR *name* is declared as ATOD or DTOA
Error 929	MODPAR: Parameter *name* not known
Error 930	MODPAR: Parameter *name* is missing
Error 931	EVAL: Parameter *name* is missing
Error 932	EVAL: Parameter *name* not known
Error 933	Error in TRISE or TFALL or TCROSS specification
Error 934	.NOISE output must be a node voltage (V(xx))
Error 935	.PBOUND: parameter *name* not found
Error 936	*name* Inconsistent FAS instance
Error 937	.LIB: keyword 'KEY =' expected: *name* found
Error 938	.MCMOD: expecting LOT or DEV keyword: *name*

Table A-8. Miscellaneous Errors

Error Number	Description
Error 939	*name* Unknown .PLOT/.PRINT specification
Error 940	.CHECKSOA: Not compatible with *name*
Error 941	*name* cannot be redefined in a .PARAM statement
Error 942	.CHRSIM *name*: Unknown argument
Error 950	Missing FPORT number *name*
Error 951	.FFILE: expecting unit (Hz, KHz, MHz, GHz): *name* found
Error 952	.FFILE: expecting format MA or RI: *name* found
Error 954	.PZ: expecting V(pin) or I(Vxx)
Error 955	Missing state *name* to be displayed for device *name*
Error 956	.TEMP: real value expected, *name* found
Error 957	.FFILE: expecting S/Y/Z: *name* found
Error 958	.ALTER inside .INCLUDE is not allowed
Error 959	.PLOT (LOW, HIGH) values must be real: *name* found
Error 960	.SETSOA: Unexpected keyword 'ELSE' found
Error 961	.SETSOA: source line *name*: Error in nesting IF-ENDIF statement
Error 962	PORT *name* has only one connection
Error 963	*name*: Invalid expression for .EXTRACT DC
Error 964	Port specification in PLOT/PRINT is larger than the number of declared PORTS
Error 965	Inconsistent number of pins for devices *name*
Error 966	M factor is less than or equal to zero for device *name*
Error 967	.LIB: .LIB KEY=*name* already exists
Error 968	Instance already defined: *name*
Error 969	Unknown MC specification *name*
Error 970	Missing MC specification in *name*
Error 971	*name*: Parameter name cannot start with a digit
Error 972	FNS order must be an integer: *name* found
Error 973	.LIB: Unable to find library or variant
Error 974	LOTGROUP *name* not defined

Table A-8. Miscellaneous Errors

Error Number	Description
Error 975	FOUR output: .H extension expected on the wave *name*
Error 976	Too many simulation are required (> ELDO_LONGVAL)
Error 977	.H extension on wave: only integer value expected: *name* found
Error 978	*name*
Error 979	IBIS: *name*
Error 980	IBIS: bad value specified for parameter *name*
Error 981	value cannot be negative
Error 982	Optimization cannot work when command *name* is active
Error 983	a non FFT output *name* appears in a FOUR or FOURMODSST extract
Error 984	unmatched if/else/then statement at or near line *name*
Error 985	Unable to alter parameter *name* because it controls device instantiation
Error 986	Entity *name* affected by both MC and a non-MC change
Error 987	.CHRISM: unexpected input format: *name*
Error 988	.NET: *name*
Error 989	At least two values are needed
Error 990	.MPRUN: *name*
Error 991	.EXTRACT mode
Error 992	Parameter *name* is not allowed
Error 993	Unable to create/alter PORT structure on elements *name* which have not been instantiated with PORT information in the netlist
Error 994	Parameter *name* should not contain boolean operator
Error 995	Parameter *name* should not contain any special characters
Error 996	.EXTRACT: wave *name* is incompatible with extract analysis
Error 997	.CORREL: need at least 2 *name* to generate a correlation
Error 998	.DSPMOD: *name*
Error 999	.DSP: Syntax error in waveform definition
Error 1000	Using a parameter both in OPTIMIZATION and STEP is not allowed
Error 1501	This circuit exhibits singularity, due to ...
Error 1502	.OPTION DSCGLOB: unexpected specification *name*

Table A-8. Miscellaneous Errors

Error Number	Description
Error 1503	.OPTION parameter DSCGLOB disabled. Use now DSCGLOB = X or DSCGLOB = GLOB
Error 1504	.STEP LIB: *name*
Error 1505	COMMAND *name*:
Error 1506	Problem in using ‘\\’ in *name*
Error 1507	Object *name*: A periodic source must be associated to PHNOISE
Error 3001	COMMAND .MEAS: duplicate measurement *name*
Error 3002	OBJECT *name*: varying parameters can’t be used with this object
Error 3003	Unable to open format *name*
Error 3004	*name* should be an INTEGER
Error 3005	nesting error inside #MACHBB directive
Error 3006	#MACHBB directive must be at top level
Error 3007	Can’t find subckt *name* for macro simulation
Error 3008	Only commands are accepted here
Error 3009	COMMAND .CALL_TCL *name*
Error 3010	FBLOCK model is not available for HP 64bits version
Error 3011	COMMAND .PARAMOPT: no initial value given

Warning Messages

Global Warnings

Table A-9. Global Warnings

Warning Number	Description
Warning 1	Mismatch between .iic and .cir files. .RESTART ignored
Warning 2	The restarting values for .RESTART DC are only usable when no DC analysis is required. DC analysis omitted
Warning 3	Unusual simulation time

Table A-9. Global Warnings

Warning Number	Description
Warning 4	Due to parameter PTF, the integration scheme has been changed from Trapeze to Backward Euler
Warning 5	unusual value for EPS *name*
Warning 6	VDS and VGS initializations are useless with Eldo
Warning 7	No output to display
Warning 8	Due to .OPTION SWITCH, the ANALOG specification is ignored
Warning 10	.RESTART: a catenated file
Warning 12	.OPTION EPSDIG ignored due to .OPTION ANALOG
Warning 13	.AC: Values for lower and upper frequencies are inverted
Warning 14	Required ‘raw’ directory does not exist in the current directory; .OPTION sda=2 ignored
Warning 15	AC analysis: *name*
Warning 16	Digital description ignored in AC analysis
Warning 17	.OPTION TIMEDIV ignored due to .OPTION SIMUDIV
Warning 18	.OPTION EPS set to *name*
Warning 19	Results from a TRAN analysis are being used for a DC analysis
Warning 20	.OPTION GRAMP accepts only positive integer values below 20 .OPTION GRAMP ignored
Warning 21	Character 'O' found just after a '.' in line *name*
Warning 22	Non invertible matrix
Warning 23	There are BJT elements in the design, LVLTIM is set to 2
Warning 24	.TOPCELL should appear before any .SUBCKT definition
Warning 25	Rounding errors may occur between the Digital Simulator and Eldo
Warning 26	NUMDGT value *name* not accepted: default value used
Warning 27	Probable syntax error in library *name*
Warning 28	Negative or zero value for *name* ignored
Warning 29	Such messages will not be displayed in future. Set .OPTION MSGNODE=0 to receive all warnings
Warning 30	Unable to open file *name*
Warning 31	No transient analysis specified .RESTART ignored

Table A-9. Global Warnings

Warning Number	Description
Warning 32	No FOUR plot to display: FFT analysis removed
Warning 33	XL: VSW0 and VSW1 set to *name*
Warning 34	UIC specification ignored on .TRAN card in ADMS
Warning 35	Multi-platform MC cannot be used
Warning 36	Unable to open LIB file *name*
Warning 37	Multiple definition of parameter *name*
Warning 38	Command *name* is ignored
Warning 39	First line of the command file must be a comment
Warning 40	.OPTION parmhier: ‘local’ or ‘global’ expected
Warning 41	Signals will be displayed in the digital output file
Warning 43	overwriting MIN/MAX of parameter *name*
Warning 44	.MP not implemented with SST analysis
Warning 45	DATA *name*
Warning 46	Double definition for parameter *name*
Warning 47	Unusual value given for *name*
Warning 48	Multiple .TEMP not allowed in ADMS: only last TEMP value will be used
Warning 49	This circuit exhibits singularity, due to *name*
Warning 50	*name* is ignored because of Mach
Warning 51	Using .OPTION RMV0, *name* zero-voltage sources would have been removed from the database
Warning 52	Optimizer not active with Eldo-demo
Warning 53	Eldo Mach partition UI can’t be used in Eldo Mach Black-Box mode
Warning 54	Eldo Mach Black-Box mode disabled because of Eldo Mach partition UI
Warning 55	.OPTION *name* has been set to

Warnings Related to Nodes

The following warning messages are preceded by `NODE <name>`:

Table A-10. Warnings Related to Nodes

Warning Number	Description
Warning 101	Significant node voltage variation, but of a lesser magnitude than VTH at time: *number*
Warning 102	Number of iterations at time: *number*
Warning 103	Attempt to redefine .IC
Warning 104	Attempt to redefine .NODESET
Warning 105	Decreasing time on time-values associated to the .IC command
Warning 106	Global node already defined
Warning 107	Less than two connections
Warning 108	This node is a floating gate
Warning 109	This bulk node is not biased
Warning 110	Connected to capacitors only
Warning 111	Less than two connections. Line: *number*
Warning 112	Connected to grounded capacitors only. This node is removed from the netlist
Warning 113	Not connected to any element. This node is removed from the netlist
Warning 114	Second input signal ignored on that node
Warning 115	This is an ATOD and DTOA node
Warning 116	STRENGTH's CONFLICT on this node. Return to DIGITAL
Warning 117	Not connected to any power supply
Warning 118	No DC path to ground
Warning 119	VTH1 > VTH2 on A2D convertor
Warning 120	VLO > VHI on D2A convertor
Warning 121	Signals will be displayed in the digital output file
Warning 122	DC dangling node
Warning 123	A capacitor of more than 1F is attached to that node
Warning 124	Appears in .CONNECT only
Warning 125	Previous IC value superseded
Warning 126	Previous NODESET/GUESS value superseded

Table A-10. Warnings Related to Nodes

Warning Number	Description
Warning 127	Possibly no DC path on that node
Warning 128	.CONNECT: node not found
Warning 129	has been replaced by a DSPF network

Warnings Related to Objects

The following warning messages are preceded by `OBJECT <name>`:

Table A-11. Warnings Related to Objects

Warning Number	Description
Warning 201	Cannot be parameterized
Warning 202	Must be a differential amplifier. AC analysis omitted
Warning 203	Object not in the linear domain. AC analysis omitted
Warning 204	Already defined
Warning 205	Unrecognized word: *name*
Warning 206	No parameter or model specified
Warning 207	Second L value ignored
Warning 208	Second W value ignored
Warning 209	Parameter ignored: *name*
Warning 210	Z0 set to 50 W
Warning 211	RON set to 1 kW
Warning 212	W is less than WMIN. W is set to WMIN
Warning 213	L is less than LMIN. L is set to LMIN
Warning 214	Value becomes negative
Warning 215	W undefined. Default value is taken: *number*
Warning 216	L undefined. Default value is taken: *number*
Warning 217	RS is negative or zero: value reset to
Warning 218	Improper value. Default value is taken (1.0×100). Value: *number*
Warning 219	Undeclared inductor
Warning 220	Default values used for time specification

Table A-11. Warnings Related to Objects

Warning Number	Description
Warning 221	Instability may occur because some of the root modules of the denominator are greater than 1, (outside the unit z circle). Poles are dumped in the standard output file
Warning 222	Instability may occur because the real parts of some of the denominator's root are greater than 1, (outside the unit z circle). Poles are dumped in the standard output file
Warning 223	Last specification ignored
Warning 224	Parameter already used as an equivalent: *name*
Warning 225	CPIN already called on pin: *name*
Warning 226	VSTATUS already called on: *name*
Warning 227	Resistor value is 0: object not called: *name*
Warning 228	This resistor is not a RC line. Parameter(s) ignored: *name*
Warning 229	WEFF=W-2DW too small or negative. Default value RES used
Warning 230	LEFF=L-2DL ≤ 0.0. Default value RES is used
Warning 231	LEFF and WEFF ≤ 0.0. Default value CAP is used
Warning 232	DTOA converter: Rhigh is not specified, the default value of 1 mW is taken
Warning 233	DTOA converter: Rlow is not specified, the default value of 1 mW is taken
Warning 234	This element cannot be converted into its switch equivalent
Warning 235	Capacitance value is negative or zero
Warning 236	Node *name* PARAM:xxxx found: This might be confused with PARAM:xxx
Warning 237	The following dimension has an unusual value: *name*
Warning 238	.SIGBUS: VHI < VLO
Warning 239	VHI < VLO on .D2A
Warning 240	Object Warning
Warning 241	Unable to plot the capacitances of this device, only charges are available
Warning 242	Unable to plot the charges of this device, only capacitances are available
Warning 243	This object makes reference to the above node, which will be assumed to be ground node 0
Warning 244	Level cannot be given on the instance
Warning 245	Self in short circuit: Object not created
Warning 246	Temperature given for TDEV device: TDEV ignored.

Table A-11. Warnings Related to Objects

Warning Number	Description
Warning 247	Capacitance value of more than 1F
Warning 248	There are several ports with that index
Warning 249	The first pin of that element is undefined: Object not created
Warning 250	The second pin of that element is undefined: Object not created
Warning 251	Old syntax: Please use Ixx or Vxx element with PORT specification
Warning 252	Self-connected objected not created.
Warning 253	Parameter PGATE is defined for JUNCAP model only
Warning 254	Differential lines are not supported, the reference plane will be replaced by the ground
Warning 255	Size not consistent with LMIN/LMAX/WMIN/WMAX
Warning 256	Very small value specified.
Warning 257	Values are out of range for
Warning 258	May be diode instantiation was intended, but its name could not start by ‘DEL’, which is a prefix reserved for Eldo DELAY primitive.
Warning 259	May be CCCS instantiation was intended, but its name could not be started by ‘/FNZ’, which is a prefix reserved for Eldo FNS/FNZ primitive.
Warning 260	May be SUBCKT instantiation was intended, but its name could not be started by ‘XOR’, which is a prefix reserved for Eldo XOR primitive.
Warning 261	May be Current Source instantiation was intended but its name could not be started by ‘INV’, which is a prefix reserved for Eldo INVERTOR primitive
Warning 262	Incorrect value for keyword ABS (1 or 0 expected). Using 0 as default
Warning 263	Ambiguity on parameter *name*
Warning 264	Object not yet created
Warning 265	W or L not specified on that device. Default setting relevant to that model will be used
Warning 266	The timestep chosen for the simulation exceeds the delay
Warning 267	Very small resistors have been encountered. This may cause non-convergence problems

Warnings Related to Commands

The following warning messages are preceded by `COMMAND <name>`:

Table A-12. Warnings Related to Commands

Warning Number	Description
Warning 301	.AC: FREQ1 is set to 1Hz
Warning 302	.AC: FREQ2 is set to 1Hz
Warning 303	.IC *name*: Ignored
Warning 304	.NODESET *name*: Ignored
Warning 305	.SOLVE: R sweep must be between positive values
Warning 306	.SOLVE: No solution has been found
Warning 307	.SOLVE: No solution has been found in the given interval
Warning 308	.IC or .NODESET *name*: This value does not match the value specified in the input signal. Zero-time value is: *number*
Warning 309	.CHRSIM: The input signal will force the simulation to end at V=*number*
Warning 310	.CHRSIM: The input signal will force the simulation to end at time = *number*
Warning 311	.OPTION *name*: Ignored
Warning 312	.OPTIMIZE *name*: Ignored
Warning 313	.WIDTH *name*: Ignored
Warning 314	.OPTFOUR: Obsolete statement ignored
Warning 315	.GLOBAL: Nodes specified in this card are considered as global nodes from the introduction of the corresponding .GLOBAL card, NOT BEFORE
Warning 316	.ENDS: The *name* does not match the .SUBCKT *name*
Warning 317	*name*: Option unknown
Warning 318	.RAMP *name*: Specification unknown. .RAMP ignored
Warning 319	.PZ: Previous specification on *name* ignored
Warning 320	.STEP: Previous card superseded
Warning 322	.NOISE: Previous specification ignored
Warning 323	.DC: Previous values superseded
Warning 324	.RAMP: Previous values superseded
Warning 325	.TRAN: Previous values superseded
Warning 326	.OPTIMIZE/.EXTRACT: Node *name* is connected to ground

Table A-12. Warnings Related to Commands

Warning Number	Description
Warning 327	.TRAN: The parameter TMAX is not used
Warning 328	.AC: Previous values superseded
Warning 329	.TRAN: Specification ignored
Warning 330	.TRAN: UIC specification accepted
Warning 331	.SENS: Element *name* not found
Warning 332	.SENS: Node *name* not found
Warning 333	.PZ: Node *name* not found
Warning 334	.PZ: Element *name* not found
Warning 335	.TF: Node *name* not found: .TF ignored
Warning 336	.TF: Element *name* not found: .TF ignored
Warning 337	.FIX: Element *name* not found
Warning 338	.UNFIX: Element *name* not found
Warning 339	.NOISE: Element/object *name* not found
Warning 340	.MC: Element *name* not found
Warning 341	.MC: Node *name* not found
Warning 342	.STEP: Parameter *name* not found
Warning 343	.STEP: No MOS *name* found
Warning 344	.OPTFOUR: Voltage source *name* found
Warning 345	.OPTFOUR: Node *name* found
Warning 346	.PRINT or .PLOT: Node *name* undeclared
Warning 347	.PRINT or .PLOT: Element *name* undeclared
Warning 348	.DC: Unknown sweep source *name*
Warning 349	.AC: No output specified in AC analysis: use .PLOT or .PRINT statement
Warning 350	.MC: Specification ignored
Warning 351	.OPTIMIZE: Ignored
Warning 352	.NOISE: Must be used in AC analysis only. NOISE analysis omitted
Warning 353	.IC: Node *name* not found
Warning 354	.NODESET: Node *name* not found
Warning 355	.GUESS: Node *name* not found

Table A-12. Warnings Related to Commands

Warning Number	Description
Warning 356	.CONSO: Voltage source *name* not found
Warning 357	.PROBE: Too many nodes to display. This value can be overridden by .OPTION LIMPROBE = value
Warning 358	.OPTIMIZE: New AC input *name* is ignored
Warning 359	.PARAM: “=” is missing in expression *name*
Warning 360	.OPTIMIZE: Unexpected expression: *name*
Warning 361	.EXTRACT: Unexpected expression: *name*
Warning 362	.OPTIMIZE: Bounds for *name* must be MIN MAX
Warning 363	.PLOT/.PRINT: Wave *name* will be plotted in .MEAS file
Warning 364	.OPTOMIZE: GAIN target is unusually high: *name*
Warning 365	.PLOTBUS: Bus *name* not defined
Warning 366	.ANALOG: *name* not found
Warning 367	.PRINT or .PLOT or .PROBE: *name* not found
Warning 368	.WCASE: *name* not known
Warning 369	.MC ignored: Neither LOT or DEV specification found
Warning 370	.SETSOA: Expecting D E or M: *name* found
Warning 371	.SETSOA: Specification *name* unknown
Warning 372	.SETSOA: Object *name* unknown
Warning 373	.SETSOA: Model *name* unknown
Warning 374	*name* analysis omitted because of the DCSWEEP analysis found
Warning 375	.SETSOA: Double specification on *name*
Warning 376	.LOOP: Object *name* undefined
Warning 377	.NOISETRAN is currently incompatible with *name*. Transient Noise Analysis is therefore omitted. For example, when a netlist contains both .MC and .NOISETRAN commands, the .NOISETRAN command is ignored.
Warning 378	EPS is possibly too large (*name*): The usual EPS value for Transient Noise is 1.0e-6
Warning 379	RELTOL is possibly too large (*name*): %e The usual RELTOL value for Transient Noise is
Warning 380	VNTOL is possibly too large (*name*): %e The usual VNTOL value for Transient Noise is
Warning 381	.PLOT/.PRINT: Wave *name* will not be plotted in .ASCII file

Table A-12. Warnings Related to Commands

Warning Number	Description
Warning 382	.OPTION *name*
Warning 392	.STEP PARAM *name*: Sweep already defined in the .DC command
Warning 393	.PROBE ISUB (*name*): No wild character allowed
Warning 394	.PZ ignored because of FNZ functions
Warning 395	.PROBE *name*: No nodes/objects found
Warning 396	ISUB(*name*) not accessible: *name* is a global node
Warning 397	ISUB: Node *name* not found
Warning 398	.PROBE: Missing keyword AC, DC or TRAN
Warning 399	Unable to display I(M/B/Jxx): *name* assumed
Warning 400	.TRAN: TSTART > TSTOP: TSTART set to 0
Warning 401	.STEP: INCR_SPEC > T2-T1
Warning 402	.NOISE: zero or negative argument
Warning 403	.AC: Frequency cannot be 0.0 or less: The value has been reset to 1.0
Warning 404	.MODDUP: element *name* not found
Warning 405	.MODDUP: cannot be used on element *name*
Warning 406	.LIB cannot find *name*
Warning 407	.SETSOA ignoring SOA on model *name*
Warning 408	.SETSOA ignoring SOA on object *name*
Warning 409	.TF analysis not performed: Bad operating point
Warning 410	The line is too long: It has been truncated
Warning 411	.OPTNOISE: previous command superseded
Warning 412	.OPTNOISE: ON/OFF expected: *name* found
Warning 413	.MC: Unknown specification *name*: LOT or DEV expected
Warning 414	.FOUR: Parameter TSTOP is ignored
Warning 415	.TRAN: Parameter TPRINT is set to: *name*
Warning 416	.OPTFOUR: Previous values superseded
Warning 417	.USE: DC value for source *name* has been changed
Warning 418	.GLOBAL: Some subcircuits are already defined, if these subcircuits make use of nodes which will appear in .GLOBAL statements, results can be in error

Table A-12. Warnings Related to Commands

Warning Number	Description
Warning 419	.OPTFOUR: Using INTERPOLATE=0 may increase the number of points computed by the simulator
Warning 420	Missing FPORT number *name*
Warning 421	LOT/DEV specification on parameter *name* ignored
Warning 422	.PZ: Specified device is not a ‘voltage-like' element
Warning 423	.OPTFOUR: TSTART > TSTOP: TSTART is ignored
Warning 424	.OPTFOUR: TSTART < 0: TSTART is ignored
Warning 425	.OPTFOUR: TSTOP < 0: TSTOP is ignored
Warning 426	.OPTFOUR: NBPT < 0: NBPT is ignored
Warning 427	.OPTFOUR: FS < 0: FS is ignored
Warning 428	.OPTFOUR: Fnormal < 0: Fnormal is ignored
Warning 429	.OPTFOUR: Fmin < 0: Fmin is ignored
Warning 430	.OPTFOUR: Fmax < 0: Fmax is ignored
Warning 431	.OPTFOUR: Fund < 0: Fund is ignored
Warning 432	*name*: Only Magnitude or DB of NOISE outputs are available
Warning 433	LOT/DEV specification is ignored on the non-primitive parameter *name*
Warning 434	The given value for VSW0 is superior to VSW1:The values will be interchanged
Warning 435	DEFAD/DEFAS/DEFPD/DEFPS overwritten by .OPTION XA for MOS models BERKELEY/BSIMxx/EKV and MM9
Warning 436	.EXTRACT: Unable to open file *name*
Warning 437	.PLOT/.PRINT: wave *name* will be plotted as
Warning 438	.TRAN: last Tstep ignored.
Warning 439	.SETBUS: last Time point ignored
Warning 440	*name* ignored with Eldo Mach
Warning 441	.STEP: increment must be positive: *name* used
Warning 442	.PARAM *name*
Warning 443	.MEAS *name*
Warning 444	.STEP: Compared to Eldo 5.4, the semantic of the .STEP ... LIN has changed: LIN stands for ‘number of runs’ Rather use keyword INCR if ‘increment value’ is meant

Table A-12. Warnings Related to Commands

Warning Number	Description
Warning 445	.TVINCLUDE: *name*
Warning 446	.PLOT: *name* statement removed. It appears more than once.
Warning 447	.STEP LIB: *name*
Warning 448	.MPRUN *name* Ignored: Cannot distribute jobs (no multi-run analysis or no hosts)
Warning 449	.NET *name*
Warning 450	EXTRACT MODE: *name*
Warning 451	Can't update correlation coefficient depending of parameter *name*
Warning 452	.CORREL ignored, no statistical analysis found.
Warning 453	.DSP *name*
Warning 454	.PLOT: unexpected output *name*
Warning 455	.PLOTBUS: limited to 53 bits: *name* not plotted
Warning 456	.SIGBUS: Only first values specified for bus *name* will be taken for .SST analysis
Warning 457	.STEP: no SST analysis found, keyword (AUTOINCR) ignored
Warning 458	.MC: vary was already specified...
Warning 459	.EXTRACT DC: DCTRAN assumed since both transient and ac analysis found in the netlist

Warnings Related to Models

The following warning messages are preceded by `MODEL <name>`:

Table A-13. Warnings Related to Models

Warning Number	Description
Warning 501	No parameter specified. Default values are used
Warning 502	GOFF less than zero
Warning 503	Access resistors for this model are created as objects
Warning 504	Unable to match reverse and forward region. BV = *number*
Warning 505	Reset parameter
Warning 508	LAMBDA value is high and may cause problems of convergence
Warning 509	KB1 is set to *number*

Table A-13. Warnings Related to Models

Warning Number	Description
Warning 510	Parameter XQC ignored
Warning 512	IBV set to *number* milliAmps. IBV has been increased in order to create a continuous diode current representation in the breakdown region.
Warning 513	Parameter CF1 set to the default value: *number*
Warning 514	Parameter CF3 set to the default value: *number*
Warning 515	Parameters specific to BSIM appear and LEVEL is neither 8 or 11
Warning 516	LOT and DEV cannot be applied on R|L|C parameters
Warning 517	Parameter MJSW must be less than 1.0. Default is used
Warning 518	Parameter MJ must be less than 1.0. Default is used
Warning 519	LOT specification is less than or equal to zero
Warning 520	DEV specification is less than or equal to zero
Warning 521	The following parameter has an unusual value:
Warning 522	DMUT parameter appears in a non-ST model
Warning 523	SUBSN=1 is not allowed on a lateral PNP
Warning 524	VTH1 > VTH2
Warning 525	VLO > VHI
Warning 526	Model Warning
Warning 527	Parameter AF has an unusual value
Warning 528	Parameter KF has an unusual value
Warning 529	Only one threshold is allowed for BIT A2D models: The average value has been taken
Warning 530	MCMOD cannot be applied on the model parameter: It can only be applied to a nominal value: MCMOD ignored
Warning 531	Sinwave function: Theta factor is negative
Warning 532	COX is ignored because TOX is given
Warning 533	The model parameters (DW/RSH) that are specific to RC WIRE have been ignored
Warning 534	Double definition for parameter(s)
Warning 535	Level 21 is now replaced by Level *name*. For future versions make sure to use that level
Warning 536	The model parameter RHO is needed for ST model

Table A-13. Warnings Related to Models

Warning Number	Description
Warning 537	*name*, parameter ignored

Warnings Related to Subcircuits

The following warning messages are preceded by `SUBCKT <name>`:

Table A-14. Warnings Related to Subcircuits

Warning Number	Description
Warning 701	Unused parameter: *name*
Warning 702	*name*: Cannot be updated with .LIB in interactive mode
Warning 703	SUBCKT *name* cannot be that of a keyword
Warning 704	This pin *name* appears more than once on definition
Warning 705	The header of this SUBCKT contains node names defined as global

Miscellaneous Warnings

Table A-15. Miscellaneous Warnings

Warning Number	Description
Warning 901	Last time ignored on input signal at line: *number*
Warning 902	*name*: Model parameter ignored
Warning 903	Too many nodes to plot (>8)
Warning 904	Previous value of *name* superseded
Warning 905	VSAT is set to: *number*
Warning 906	VSATM is set to: *number*
Warning 907	Parameter not used: *name*
Warning 908	*name*: This parameter cannot be assigned
Warning 909	*name*: Unknown parameter
Warning 910	Unable to measure ISUB(*name*): Multiple connections
Warning 911	PORT *name* has only one connection

Table A-15. Miscellaneous Warnings

Warning Number	Description
Warning 912	IBIS: *name*
Warning 913	IBIS: parameter *name* ignored
Warning 914	IBIS: parameter *name* not in the interval [0,1] is ignored
Warning 915	Probably missing a path to ground from object *name*
Warning 916	Unable to parse expression *name*
Warning 917	SWEEP specification on TRAN/DC/AC ignored when .STEP are used: these two syntaxes are not compatible
Warning 918	The check for instance duplicates has been disabled (netlist too large). Use .OPTION CHECKDUPL to override this limit.
Warning 919	.OPTION ACCOUT=1 can not be applied to FOUR(*name*).
Warning 920	Instances *name*, * not created because M is 0*
Warning 921	MC variation on *name*

Table A-16. Miscellaneous Warnings

Warning Number	Description
Warning 1001	.NOISETRAN ignored: Only single TRAN analysis is allowed
Warning 1002	.NOISE ignored because the output node is not defined
Warning 1003	.SWITCH: Element *name* not found
Warning 1004	.MCMOD: Parameter *name* not known
Warning 1005	.MCMOD: Element *name* not known
Warning 1006	COMMAND *name* removed from the include file
Warning 1007	.SAVE and .RESTART used on the same file: .SAVE ignored
Warning 1008	COMMAND *name* found in include file
Warning 1009	.OPTION: Incorrect range for *name*
Warning 1010	.PRINT or .PLOT: AC-like output *name* for non-AC analysis
Warning 1011	.PRINT or .PLOT: Unexpected AC output *name*
Warning 1012	.PLOT element *name* is not declared
Warning 1013	.PROBE/.SAVE element *name* is not declared
Warning 1014	.VIEW element *name* is not declared

Table A-16. Miscellaneous Warnings

Warning Number	**Description**
Warning 1015	Element *name* is not found
Warning 1016	.SOLVE: Previous specifications superseded
Warning 1017	AC input sources used in connection with FPORT
Warning 1018	Digital gates are not handled in DCSWEEP
Warning 1019	.WC and .MC cannot be used together: .WC ignored
Warning 1020	.PZ analysis might give unexpected results because several AC sources are found in the circuit
Warning 1021	.PRINT or .PLOT: Unexpected AC output *name* for SST
Warning 1022	.PRINT or .PLOT: Only V() or I() is allowed for SST type
Warning 1023	.TDEV: Object *name* not found
Warning 1024	.PLOT element *name* not declared
Warning 1025	.PRINT or .PLOT: Unexpected FFT output *name*
Warning 1026	.PRINT/.PLOT/.PROBE: Missing declaration .FOUR LABEL=*name*
Warning 1027	.PRINT or .PLOT SSTNOISE: Expected SSTNOISE output, and not *name*
Warning 1028	.MEAS: Element *name* not declared
Warning 1029	.EXTRACT: Element *name* not declared
Warning 1030	.PRINT: Smith chart expects non-AC wave names
Warning 1031	.SSTPROBE *name*: second node assumed to be 0
Warning 1032	.PRINT/PLOT/PROBE/EXTRACT: bad number of tones specified
Warning 1033	.PLOT: *name*. wave type is incompatible with extract type
Warning 1034	*name*: Group-Delay specifications ignored in SST analysis
Warning 1035	.PRINT or .PLOT: Only WOPT() is allowed for OPT type
Warning 1036	SUBCKT *name* will be simulated as top level
Warning 1037	Standard Eldo Mach commands ignored in Eldo Mach Black-Box mode
Warning 1038	Command .EXTRACT OPMODE on *name*: *name* is not a valid keyword.
Warning 1039	COMMAND .STEP: keyword (AUTOINCR) is ignored. It can only be applied on .STEP PARAM syntax.
Warning 1040	COMMAND .MC: keyword OUTER is disabled. No multi-run analysis has been specified.

Table A-16. Miscellaneous Warnings

Warning Number	Description
Warning 1041	COMMAND .PRINT or .PLOT SSTJITTER: Expected SSTJITTER output, and not *name*
Warning 1042	COMMAND .PRINT or .PLOT SSTSTABIL: Expected SSTSTABIL output, and not *name*
Warning 1043	COMMAND .PRINT or .PLOT: output *name* will be written as real/imaginary parts in files which do not support complex format.
Warning 1044	COMMAND .EXTRACT or .MEAS: Unexpected *name* output
Warning 1045	COMMAND .PRINTFILE: Element *name* not declared
Warning 1046	COMMAND .DSPF_INCLUDE: file *name*. An extra X has been added on *name*. That rule of adding an extra X will now be used by default in that file for all subsequent *|NET instructions.
Warning 1047	BUS *name* is of size 0: Could be declaration of that bus is missing.
Warning 1048	.OPTION *name*: list of values are limited to 10 elements.
Warning 1049	COMMAND .OPTWIND *name* is incompatible with options defined as a list of values. Only first value is kept.
Warning 1050	COMMAND *name*:Node *name* not found
Warning 1051	COMMAND *name*:Device *name* not found
Warning 1052	Object *name*: Eldo will simulate as if CTYPE = 1 had been set on that instance... This is because C value seems not to be dependant on the pin bias, or pins are connected to power supply or ground, and in such situations, CTYPE = 1 mode is usually more appropriate.. However, in case this tuning is here not appropriate, just set CTYPE = 0 on the device, or use .OPTION NOAUTOCTYPE in order to disable this setup.
Warning 1053	COMMAND .EXTRACT: *name* is ignored.
Warning 1054	COMMAND .FOUR: *name* removed
Warning 1055	COMMAND .MC: incorrect value for PRINT_EXTRACT. *name* found while only NOMINAL, NOMLAST, ALL and numbers are allowed
Warning 1056	COMMAND .MC: Several .MC commands found in the netlist. Only last one will be used.
Warning 1057	COMMAND .PUNCH: unimplemented analyze type (*name*) requested. Command ignored
Warning 1058	COMMAND *name*: SNF output required but no .SNF card has been given
Warning 1059	COMMAND .OPTIMIZE: Type of design objective *name* is incompatible with method

Table A-16. Miscellaneous Warnings

Warning Number	Description
Warning 1060	D2A/A2D command ignored using ADMS.
Warning 1061	The SUBCKT(S) listed above will not be changed by the STEP command, they will remain the same for the entire stepping process. Therefore, results might not be as expected... Suggest to use .ALTER command in .cir file instead.
Warning 1062	COMMAND .AGE: TSTART/TSTOP specifications are ignored because of TWINDOW.
Warning 1063	COMMAND .EXTRACT/.MEAS *name*: Unable to find any .DATA associated with current analysis. Command ignored.
Warning 1064	COMMAND .EXTRACT/.MEAS *name*: Unable to find an item named *name* in .DATA *name*. Command ignored.
Warning 1065	COMMAND .EXTRACT/.MEAS *name*: Complex measurements are not compatible with optimization. Command ignored.
Warning 1066	COMMAND .EXTRACT/.MEAS *name*: Unable to find a .DATA named *name*. Command ignored.
Warning 1067	COMMAND .EXTRACT/.MEAS *name*: in *name* some points are outside the simulation interval. They will be ignored by the fitting command.
Warning 1068	COMMAND .EXTRACT/.MEAS *name*: in *name* some points are not correctly ordered, or not unique for a given X value. Fitting command ignored.
Warning 1069	COMMAND .EXTRACT/.MEAS *name*: in *name* some points can't be simulated. Fitting command ignored.
Warning 1070	COMMAND .PLOT/.PRINT/.PROBE: wave *name* contains some errors. This command is ignored.
Warning 1071	COMMAND .PLOT/.PRINT/.PROBE: Unexpected character '*name*' found
Warning 1072	COMMAND .PRINTFILE: xaxis START value for file *name* has been set to:
Warning 1073	COMMAND .PRINTFILE: xaxis STOP value for file *name* has been set to:
Warning 1074	In corner *name* of library *name* .BIND commands can't be processed by the STEP command.They will be ignored during the stepping process, therefore results might not be as expected. Suggest to use .ALTER command instead.
Warning 1075	COMMAND .RAMP: syntax error in command declaration:
Warning 1076	dc voltage reset to initial transient source value for source *name*
Warning 1077	COMMAND .PLOTBUS: incorrect BASE value: *name*. Default value will be used.
Warning 1078	COMMAND .PLOTBUS: BIN, DEC, OCT or HEX is expected after keyword BASE. Default value will be used.

Table A-16. Miscellaneous Warnings

Warning Number	Description
Warning 1079	COMMAND .PLOTBUS: UNSIGNED or 2COMP is expected after keyword RADIX. Default value will be used.
Warning 1080	COMMAND .PLOTBUS: display keywords BASE and RADIX are mutually exclusive. Priority is given to RADIX.
Warning 1081	COMMAND .PLOTBUS: display keywords BASE and RADIX are ignored. They do not apply on analog buses.
Warning 1082	COMMAND .USE_TCL: NATIVE_TCL or EZWAVE_UDF is expected after keyword MODE. Default value will be used.
Warning 1083	COMMAND .TRAN is ignored when .MODSST is set.
Warning 1084	COMMAND .OBJECTIVE: unsupported analysis: *name*.
Warning 1085	COMMAND .PRINTFILE: file *name* cannot support more than *name* analysis.
Warning 1086	COMMAND .SIGBUS: Last value for bus *name* is not consistent with periodic definition. It should be equal to first value.
Warning 1087	COMMAND .OPTION: *name* has been deprecated and will be removed from future releases.
Warning 1088	COMMAND .ADMS_RESTART: argument evaluates to 0.0analog simulation start will not be delayed
Warning 1089	COMMAND .OPTIMIZE: .STEP command containing parameter *name* specified in OUTER_PARAM contains other parameters. OUTER_PARAM will affect them all.
Warning 1090	COMMAND .SCALE: Unable to find a device or model parameter to scale
Warning 1091	COMMAND .OPTFOUR: FMIN/FMAX specifications can not be applied on RF signals
Warning 1092	COMMAND .BIND cannot be used for FS Black Box Instances *name*.
Warning 1093	COMMAND .LIMIT: complex output is not allowed inside *name*. Statement ignored.
Warning 1094	COMMAND .MEAS: duplicate measurement name *name*
Warning 1095	COMMAND .MODEL *name*: parameter *name* has been deprecated and will be removed from future releases.
Warning 1096	COMMAND .MODEL *name*: parameter *name* ignored.
Warning 1097	COMMAND .extract or .meas *name* contains symbols which are not handled in extract-mode. This command is ignored.
Warning 1098	COMMAND .FOUR: *name* removed. Node *name* is unknown.

Table A-16. Miscellaneous Warnings

Warning Number	Description
Warning 1099	COMMAND .FOUR: *name* removed. Device *name* is unknown.

Appendix B
Troubleshooting

Introduction

Most users of any product will from time to time experience problems in its use. This appendix is intended to provide an easy to use quick reference from which such problems can be identified and corrected.

Common Netlist Errors

Below is a list of the common errors to look out for when producing an input netlist:

- First Line of a File Not the Title

  The first line of a netlist *must* be the title of the circuit. If part of the circuit description is found on the first line, it will be ignored. If the first line is a blank, this will be taken as the title line.

- Correct Usage of the Simulator Accuracy (`eps`, `vntol`, `reltol`)

  The simulator accuracy parameters are the most important Eldo parameters—they must be correctly initialized.

For more details refer to the Speed and Accuracy chapter.

- Correct Units on Devices

  It is very important to make sure that all the components in the netlist have correct units; one mistake can have a large effect. An example of this is specifying units of farads instead of picofarads.

- Missing Model

  Make sure that model definitions are available to the simulator, either directly in the netlist, or in a referenced library.

- Missing Voltage Sources

  Independent voltage sources defined to have a voltage of 0 V may be removed by Eldo in order to simplify calculations. If you wish to use a voltage source as a current probe, avoid defining its voltage as 0 V, but instead, define its voltage value to be very small. Moreover, currents through components may be measured directly, therefore, the use of voltage sources as current probes is not necessary.

- Correct Model Name Syntax

  The first character of a model name *cannot* be a number. This causes the compilation of the netlist to be interrupted, and an error message to be displayed.

- Unknown Model Parameter

  When defining model parameter values, care must be taken to ensure that the parameter is spelt correctly and that it is indeed a legal parameter of the device.

- Ground (0) Node

  A 0 node (i.e. ground) must always be present in the input netlist.

Note

Eldo does not recognize GND as GROUND!

- Reserved Keywords

  Some keywords should not be specified in a **`.PARAM`** command. If they are then errors will be generated. See Table 10-21 and Table 10-22 on page 779.

Appendix C
Examples

Introduction

This chapter contains the set of examples that are distributed with the Eldo software package. Each example consists of the complete Eldo netlist of the circuit, together with output results obtained from EZwave, the Eldo waveform viewer. A summary of the examples is shown below. Listings for these examples may be found in the following subdirectories included with your software:

```
$MGC_AMS_HOME/examples/eldo/
```

where `$MGC_AMS_HOME` is the directory where the software resides.

Note

For more examples please refer to "Examples for IEM" on page 1102 and the Tutorials chapter.

Table C-1. Eldo Examples

Example	Circuit Name
Example#1—SC Schmitt Trigger	*trigger.cir*
Example#2—4-bit Adder	*ad4b.cir*
Example#3—CMOS Op-amp (Open Loop)	*aopalt.cir*
Example#4—CMOS Op-amp (Closed Loop)	*aopbou.cir*
Example#5—5th Order Elliptic SC Low Pass Filter	*ellipt5.cir*
Example#6—Charge Control in MOS 4 and 6	*niv4_6.cir*
Example#7—Active RC Band Pass Filter	*ua741.cir*
Example#8—2nd Order Delta Sigma Modulator	*integrator.cir*
Example#9—Operational Amplifier	*extract.cir*

Example#1—SC Schmitt Trigger

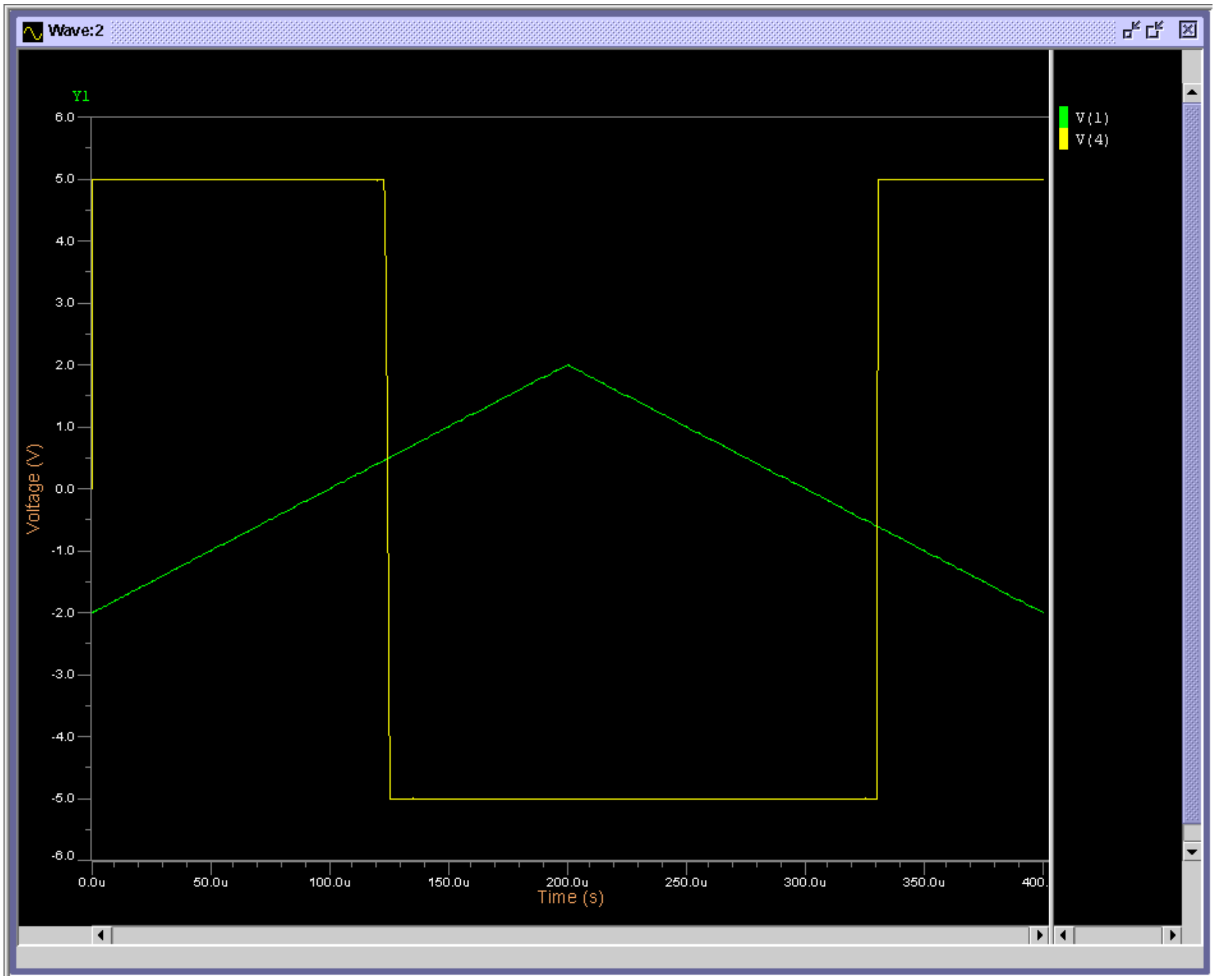
This example deals with a transient analysis of an SC Schmitt trigger circuit. The complete netlist can be found in the file *trigger.cir*.

Complete Netlist

```
Schmitt trigger
.model ampop opa level=2 voff=0 sl=50e06
+ cin=0 rs=10 vsat=5 gain=5000 fc=5000
.subckt ampli 1 2 3
opa1 2 1 3 0 ampop
.ends ampli
s1 cl  1 2 1k
s2 cl  3 6 1k
s3 clb 6 8 1k
s4 clb 9 4 1k
s5 cl  9 0 1k
c1 2 0 0.01n
c2 3 0 0.01n
c3 6 8 0.01n
c4 8 9 0.001n
x1 2 3 4 ampli
x2 8 0 6 ampli
.chrent cl 0 -5 0.1u 5 4.9u 5 5u -5 10u -5 p
.chrent clb 0 -5 5u -5 5.1u 5 9.9u 5 10u -5 p
.chrent 1 0 -2 0.2m 2 0.4m -2 f
.tran 1u 0.4m uic
.plot tran v(1) v(4) (-6,6)
.print tran v(1) v(4)
.option eps=1e-4
.end
```

Simulation Results

Figure C-1. Example#1—Simulation Results



Example#2—4-bit Adder

This example deals with a transient analysis of a 4-bit adder. The complete netlist can be found in the file *ad4b.cir*.

Complete Netlist

```
ad4b
.model mod1 nmos
+ niv=6 eox=25.0N muo=600 nb=2.0e+16 dphif=0.6 vl=2.0e5
+ kw=2.24U kl=2.24U gw=3.91U gl=0.7U dinf=0.1 ve=10.0e+4
+ ldif=10U cdifs0=0.0001 cdifp0=0 dw=0 dl=0.8u rec=0.15u
+ vt0=0.55 kb=0.1 tg=0.06
```

```
.model mod2 pmos
+ niv=6 eox=25n muo=225 nb=2.0e+16 dphif=0.7 vl=1.9e+5
+ kw=0.52u kl=4u gw=0.453u gl=0.7u dinf=0.17 ve=7.35e+4
+ ldif=10u cdifs0=0.000316 cdifp0=0 dw=0 dl=1.5u
+ rec=0.5u vt0=0.55 kb=0.34 tg=0.14
.model mod3 nmos
+ niv=6 dw=0.5u dl=1.1u eox=60n dphif=0.7 vt0=-4.0 muo=446.0
+ kb=0.4 kw=0.0u kl=0.0u gw=2.4u gl=0.5u dinf=0.14 ve=12.8e+4
+ rec=0.4u tg=0.05 vl=1.0e+5 sh=0.1 nb=1e+15
.model mod4 nmos
+ niv=6 dw=0.5u dl=1.1u eox=60.0n dphif=0.7 vt0=0.7 muo=672.0
+ kb=0.234 kw=2.17u kl=0.49u gw=4.325u gl=0.872u dinf=0.105
+ ve=7.71e+4 rec=0.4u tg=0.05 vl=8.55e+4 sh=0.1 nb=1.0e+15
*subcircuit definition
.subckt flipflop vdd vss h d q qb
m1  n1  n2 0  vss mod4 w=15u  l=3.5u
m2  n2  n1 0  vss mod4 w=15u  l=3.5u
m3  n2  h  0  vss mod4 w=15u  l=3.5u
m4  n4  n2 0  vss mod4 w=15u  l=3.5u
m5  n1  n5 0  vss mod4 w=15u  l=3.5u
m6  n4  h  0  vss mod4 w=15u  l=3.5u
m7  n4  n5 0  vss mod4 w=15u  l=3.5u
m8  n5  n4 0  vss mod4 w=15u  l=3.5u
m9  n5  d  0  vss mod4 w=15u  l=3.5u
m10 q   n2 0  vss mod4 w=15u  l=3.5u
m11 q   qb 0  vss mod4 w=15u  l=3.5u
m12 qb  q  0  vss mod4 w=15u  l=3.5u
m13 qb  n4 0  vss mod4 w=15u  l=3.5u
m14 vdd n1 n1 vss mod3 w=6.0u l=5.0u
m15 vdd n2 n2 vss mod3 w=6.0u l=5.0u
m16 vdd n4 n4 vss mod3 w=6.0u l=5.0u
m17 vdd n5 n5  vss  mod3   w=6.0u l=5.0u
m18 vdd q  q   vss  mod3   w=6.0u l=5.0u
m19 vdd qb vss mod3 w=6.0u l=5.0u
c1 n1 0 0.029p
c2 n2 0 0.048p
c3 n4 0 0.044p
c4 n5 0 0.046p
c5 q  0 0.066p
c6 qb 0 0.053p
.ends flipflop
*subcircuit definition
.subckt adder vdd vss a b er s sr
m1  n12 b   a   vss mod4 w=4.0u  l=4.0u
m2  s   n13 n14 vss mod4 w=4.0u  l=4.0u
m3  s   er  n12 vss mod4 w=4.0u  l=4.0u
m4  n12 n16 n11 vss mod4 w=4.0u  l=4.0u
m5  n11 a   0   vss mod4 w=15.0u l=3.5u
m6  n16 b   0   vss mod4 w=15.0u l=3.5u
m7  n14 n12 0   vss mod4 w=15.0u l=3.5u
m8  n13 er  0   vss mod4 w=15.0u l=3.5u
m9  n17 b   0   vss mod4 w=15.0u l=3.5u
m10 n17 a   0   vss mod4 w=15.0u l=3.5u
```

```
m11 n15 er  0   vss mod4 w=15.0u l=3.5u
m12 n15 n12 0   vss mod4 w=15.0u l=3.5u
m13 sr  n15 0   vss mod4 w=15.0u l=3.5u
m14 sr  n17 0   vss mod4 w=15.0u l=3.5u
m15 vdd n11 n11 vss mod3 w=6.0u  l=5.0u
m16 vdd n16 n16 vss mod3 w=6.0u  l=5.0u
m17 vdd n14 n14 vss mod3 w=6.0u  l=5.0u
m18 vdd n13 n13 vss mod3 w=6.0u  l=5.0u
m19 vdd n15 n15 vss mod3 w=6.0u  l=5.0u
m20 vdd n17 n17 vss mod3 w=6.0u  l=5.0u
m21 vdd Sr  sr  vss mod3 w=6.0u  l=5.0u
c1 s   0 0.031p
c2 n11 0 0.31p
c3 n12 0 0.001p
c4 n13 0 0.018p
c5 n14 0 0.024p
c6 n15 0 0.033p
c7 n16 0 0.018p
c8 n17 0 0.036p
c9 sr  0 0.040p
.ends adder
*subcircuit calls
xa1 vdd vss n1 n4  0   n3  n5  adder
xa2 vdd vss a1 n7  n5  n6  n8  adder
xa3 vdd vss a2 n12 n8  n11 n13 adder
xb1 vdd vss h  n3  n4  S1  flipflop
xb2 vdd vss h  n6  n7  S2  flipflop
xb3 vdd vss h  n11 n12 S3  flipflop
xb4 vdd vss h  n13 n1  S4  flipflop
c1  n3  0  0.07p
c2  n4  0  0.04p
c3  n6  0  0.07p
c4  n7  0  0.04p
c5  n5  0  0.01p
c6  n8  0  0.01p
c7  n13 0  0.01p
c8  n1  0  0.04p
c9  n11 0  0.07p
c10 n12 0  0.04p
*voltage commands
.chrent  a1  0 5    f
.chrent  a2  0 5    f
.chrent  h   0 5n   5 15n 5 20n 0 30n 0 p
.chrent  vdd 0 5    f
.chrent  vss 0 -2.5 f
*output commands
.plot  tran  v(h)
.plot  tran  v(s1)
.plot  tran  v(s2)
.plot  tran  v(s3)
.plot  tran  v(s4)
.plot  tran  v(n6)
.plot  tran  v(n11)
.plot  tran  v(n13)
.tran 1ns 500ns uic
.option eps=1.0e-2
.end
```

Simulation Results

Figure C-2. Example#2—Simulation Results—1

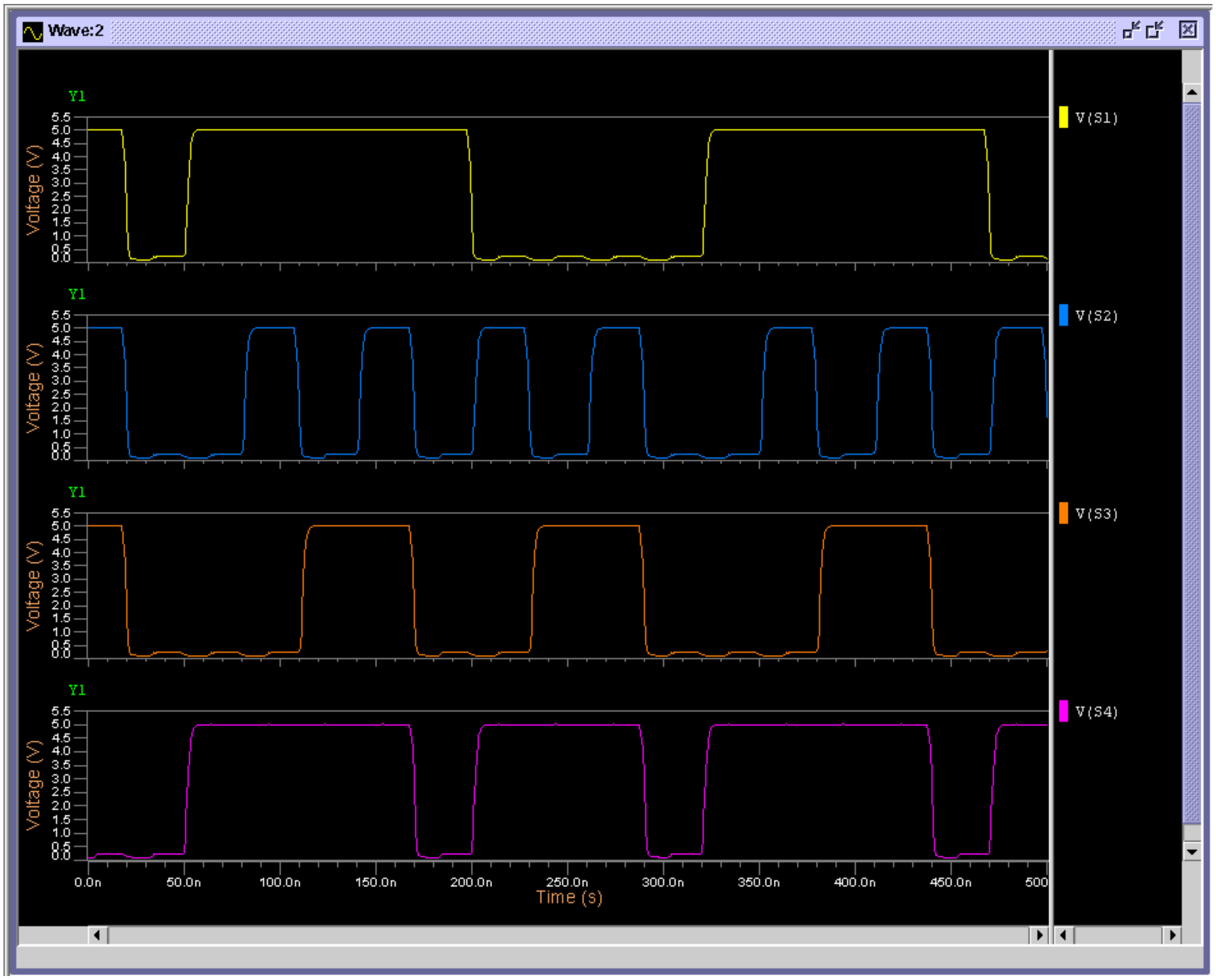
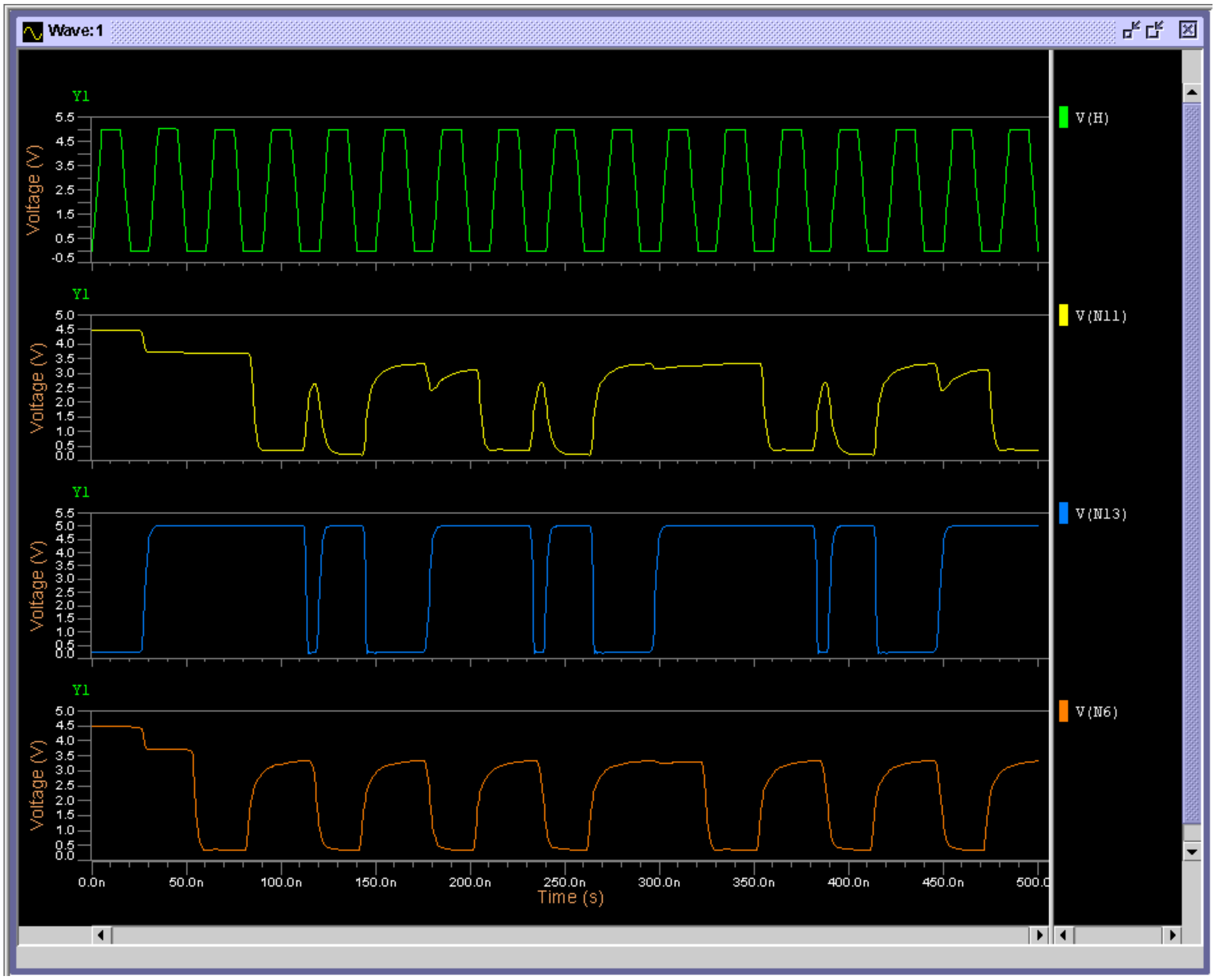


Figure C-3. Example#2—Simulation Results—2



Example#3—CMOS Op-amp (Open Loop)

This example deals with an AC analysis of an open loop CMOS operational amplifier circuit. The complete netlist can be found in the file *aopalt.cir*.

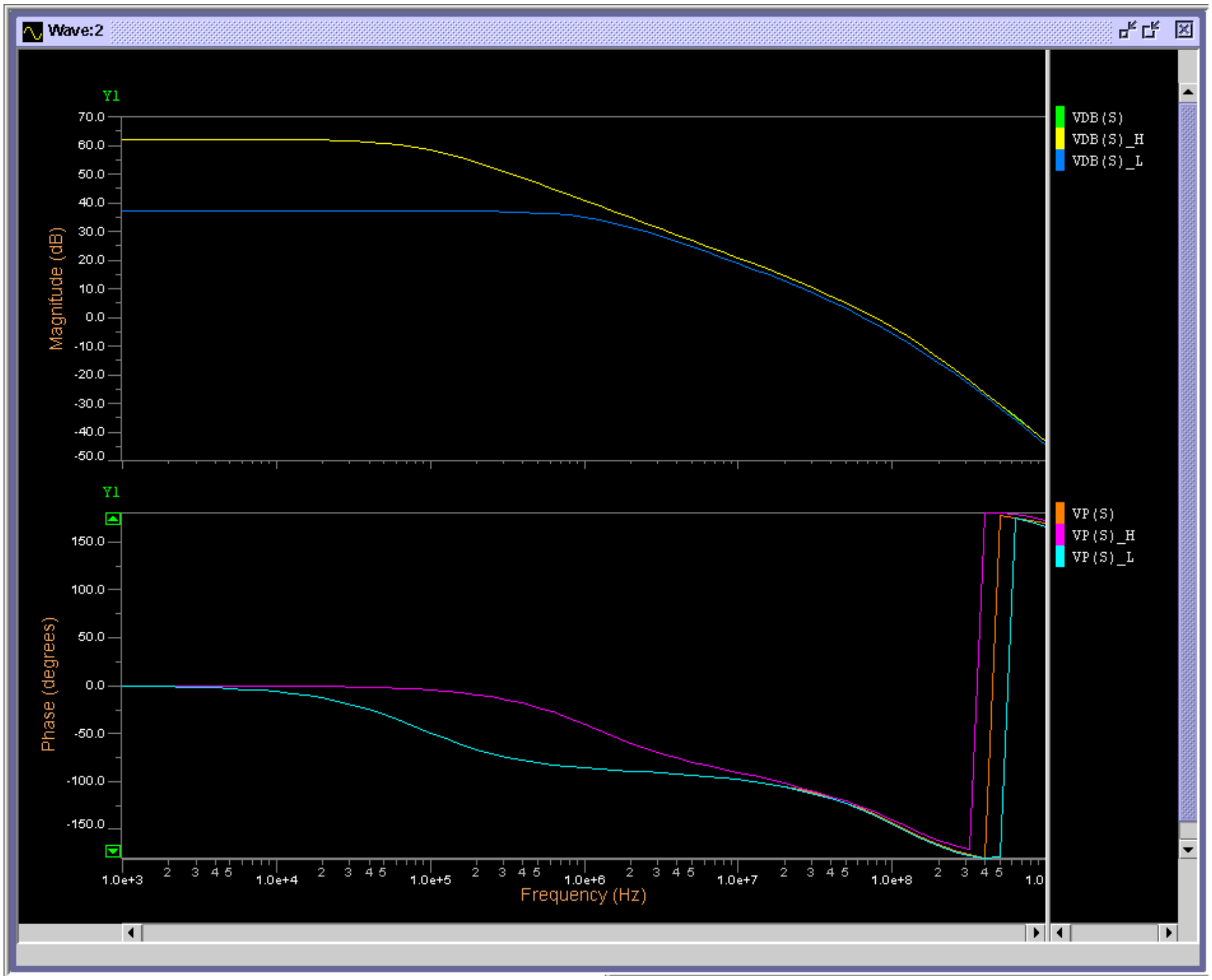
Complete Netlist

```
aopalt
.model mod1 nmos level=merck2 eox=25.0n mu0=600 nb=2.0e+16
+ dphif=0.6 vl=2.0e+5 kw=2.24u kl=2.24u lot=0.5% gw=3.91u gl=0.7u
+ dev=0.07e-6 dinf=0.1
+ ve=10.0e+4 ldif=10u  cdifs0=0.0001 cdifp0=0.0
+ dw=0.0  dl=0.8u  rec=0.15u vt0=0.55 kb=0.1 tg=0.06

.model mod2 pmos level=merck2 eox=25n mu0=225 nb=2.0e+16
+ dphif=0.7 vl=1.9e+5  kw=0.52u  kl=4u gw=0.453u gl=0.7u dinf=0.17
+ ve=7.35e+4 ldif=10u  cdifs0=0.000316 cdifp0=0
+ dw=0 dl=1.5u rec=0.5u vt0=-0.55 kb=0.34 tg=0.14
*amplifier
```

```
m1 a g vdd vdd mod2 w=120u l=5.5u
m2 b g vdd vdd mod2 w=120u l=5.5u
m3 d k a vdd mod2 w=116u l=3.5u
m4 s k b vdd mod2 w=116u l=3.5u
m5 c i vss vss mod1 w=63u l=6u
m6 a ep c vss mod1 w=130u l=4u
m7 b en c vss mod1 w=130u l=4u
m8 d d ff vss mod1 w=5.5u l=4.5u
m9 s d e vss mod1 w=5.5u l=4.5u
m10 ff e vss vss mod1 w=42u l=4u
m11 e e vss vss mod1 w=42u l=4u
m12 g g vdd vdd mod2 w=14.5u l=5.5u
m13 g g h vss mod1 w=9u l=5.5u
m14 i i h vdd mod2 w=19u l=4.5u
m15 i i vss vss mod1 w=6u l=6u
m16 j g vdd vdd mod2 w=20u l=5.5u
m17 j j k vss mod1 w=26u l=3.5u
m18 nl i k vdd mod2 w=3u l=3.5u
m19 nl nl vss vss mod1 w=4u l=3.5u
c1 s 0 1.5p
v1 vdd 0 3
v2 vss 0 -3
vinm en 0 0
vinp ep 0 ac

.mc 7 vdb(s)
.ac dec 10 1.0e3 1.0e9
.plot ac vdb(s)
.plot ac vp(s)
.option eps=1.0e-4
.end
```

Simulation Results

Figure C-4. Example#3—Simulation Results



Example#4—CMOS Op-amp (Closed Loop)

The circuit in this example is the same as in the last one with the exception being that the circuit is in a closed loop configuration. A transient analysis is performed and the complete netlist can be found in the file *aopbou.cir*.

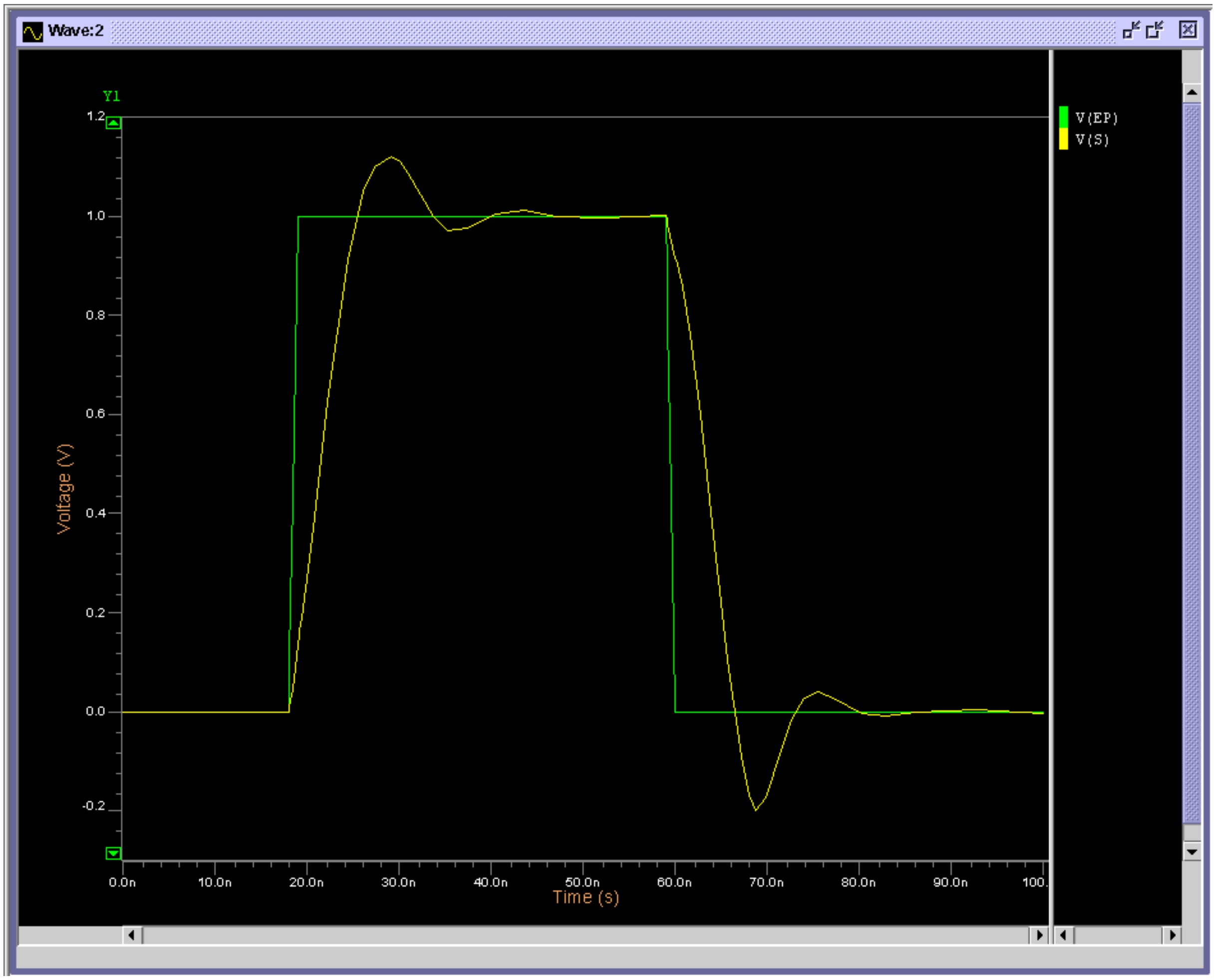
Complete Netlist

```
aopbau
.model mod1 nmos level=merck2 eox=25.0n mu0=600 nb=2.0e+16
+ dphif=0.6 vl=2.0e+5 kw=2.24u kl=2.24u  gw=3.91u gl=0.7u dinf=0.1
+ ve=10.0e+4 ldif=10u  cdifs0=0.0001 cdifp0=0.0
+ dw=0.0  dl=0.8u  rec=0.15u vt0=0.55 kb=0.1 tg=0.06

.model mod2 pmos level=merck2 eox=25n mu0=225 nb=2.0e+16
```

```
+ dphif=0.7 vl=1.9e+5  kw=0.52u  kl=4u gw=0.453u gl=0.7u dinf=0.17
+ ve=7.35e+4 ldif=10u  cdifs0=0.000316 cdifp0=0
+ dw=0 dl=1.5u rec=0.5u vt0=-0.55 kb=0.34 tg=0.14

*amplifier
m1 a g vdd vdd mod2 w=120u l=5.5u
m2 b g vdd vdd mod2 w=120u l=5.5u
m3 d k a vdd mod2 w=116u l=3.5u
m4 s k b vdd mod2 w=116u l=3.5u
m5 c i vss vss mod1 w=63u l=6u
m6 a ep c vss mod1 w=130u l=4u
m7 b s c vss mod1 w=130u l=4u
m8 d d ff vss mod1 w=5.5u l=4.5u
m9 s d e vss mod1 w=5.5u l=4.5u
m10 ff e vss vss mod1 w=42u l=4u
m11 e e vss vss mod1 w=42u l=4u
m12 g g vdd vdd mod2 w=14.5u l=5.5u
m13 g g h vss mod1 w=9u l=5.5u
m14 i i h vdd mod2 w=19u l=4.5u
m15 i i vss vss mod1 w=6u l=6u
m16 j g vdd vdd mod2 w=20u l=5.5u
m17 j j k vss mod1 w=26u l=3.5u
m18 nl i k vdd mod2 w=3u l=3.5u
m19 nl nl vss vss mod1 w=4u l=3.5u
c1 s 0 1.5p

v1 vdd 0 3
v2 vss 0 -3
v3 ep 0 pwl(0n 0 18n 0 19n 1 59n 1 60n 0)
.tran 1n 100n
.options eps=1.0e-6
.plot tran v(ep) v(s) (-0.3,1.2)
.end
```

Simulation Results

Figure C-5. Example#4—Simulation Results



Example#5—5th Order Elliptic SC Low Pass Filter

This example deals with a transient analysis on a 5th order SC low pass filter This type of circuit is used in those applications requiring the analysis of sampled data in analog circuits, being used in certain ADC/DAC and switch capacitor filters designs. The complete netlist can be found in the file *ellipt5.cir*.

Complete Netlist

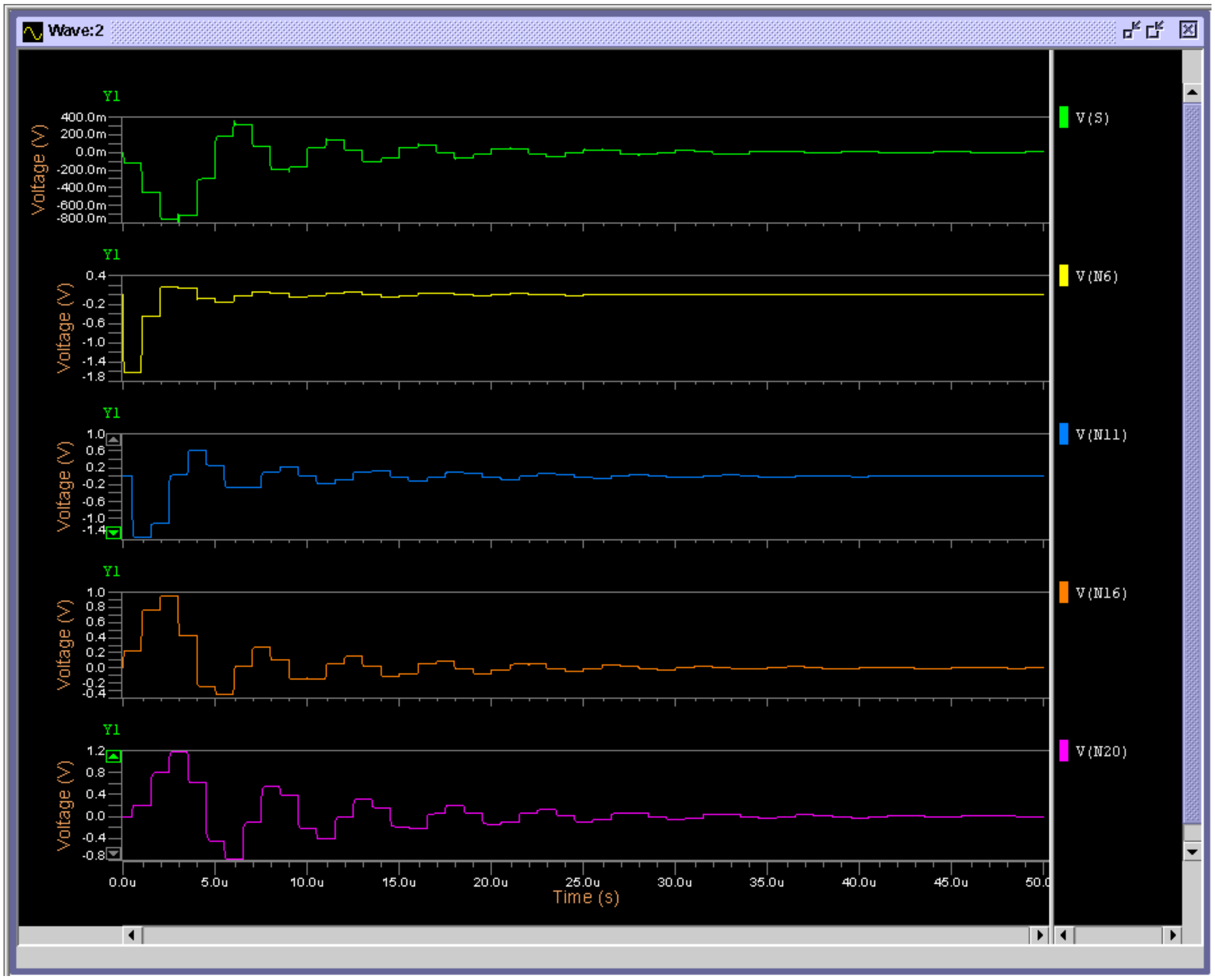
```
ellipt5
.model ampop opa level=2 voff=0 sl=50e06 cin=0p
+ rs=1 vsat=5 gain=10000 fc=5e3 cmrr=0
```

```
opa1   0  n5   n6  0    ampop
opa2   0  n10  n11 0    ampop
opa3   0  n15  n16 0    ampop
opa4   0  n21  n20 0    ampop
opa5   0  n25  s   0    ampop
sint1  ph1 e   n1  1.0k 0.0p
sint2  ph1 n11 n3  1.0k 0.0p
sint3  ph1 n4  n6  1.0k 0.0p
sint4  ph1 n2  n5  1.0k 0.0p
sint5  ph1 n6  n7  1.0k 0.0p
sint6  ph1 n16 n9  1.0k 0.0p
sint7  ph1 n8  0   1.0k 0.0p
sint8  ph1 n11 n12 1.0k 0.0p
sint9  ph1 n20 n13 1.0k 0.0p
sint10 ph1 n14 n15 1.0k 0.0p
sint11 ph1 n16 n17 1.0k 0.0p
sint12 ph1 s   n18 1.0k 0.0p
sint13 ph1 n19 0   1.0k 0.0p
sint14 ph1 n20 n22 1.0k 0.0p
sint15 ph1 S   n23 1.0k 0.0p
sint16 ph1 n24 n25 1.0k 0.0p
sint17 ph2 n1  0   1.0k 0.0p
sint18 ph2 n3  0   1.0k 0.0p
sint19 ph2 n4  0   1.0k 0.0p
sint20 ph2 n2  0   1.0k 0.0p
sint21 ph2 n7  0   1.0k 0.0p
sint22 ph2 n9  0   1.0k 0.0p
sint23 ph2 n8  n10 1.0k 0.0p
sint24 ph2 n12 0   1.0k 0.0p
sint25 ph2 n13 0   1.0k 0.0p
sint26 ph2 n14 0   1.0k 0.0p
sint27 ph2 n17 0   1.0k 0.0p
sint28 ph2 n18 0   1.0k 0.0p
sint29 ph2 n19 n21 1.0k 0.0p
sint30 ph2 n22 0   1.0k 0.0p
sint31 ph2 n23 0   1.0k 0.0p
sint32 ph2 n24 0   1.0k 0.0p
```

```
c1  n6  n5  1.50536963p
c2  n6  n15 0.2589059p
c3  n4  n2  1.0p
c4  n3  n2  1.0p
c5  n1  n2  1.0p
c6  n10 n11 0.96577222p
c7  n7  n8  1.0p
c8  n9  n8  1.0p
c9  n16 n15 2.3362151p
c10 n16 n5  0.2589059p
c11 n16 n25 0.94246071p
c12 n13 n14 1.0p
c13 n12 n14 1.0p
c14 n20 n21 0.51354696p
c15 n18 n19 1.0p
c16 n17 n19 1.0p
c17 s   n25 0.75440758p
c18 s   n15 0.94246071p
c19 n22 n24 1.0p
c20 n23 n24 1.0p
.chrent e 0n 4 700n 4 800n 0 f
.chrent phi 0.0n -5.0 10.0n 5.0 490.0n 5.0
+ 500.0n -5.0 1000.0n -5 p
.chrent ph2 0.0n -5.0 500.0n -5.0 510.0n 5.0
+ 990.0n 5.0 +1000.0n -5 p
.plot tran v(s)   (-0.8, 0.4)
.plot tran v(n6)  (-1.8, 0.4)
.plot tran v(n11) (-1.5, 1.0)
.plot tran v(n16) (-0.4, 1.0)
.plot tran v(n20) (-0.8, 1.2)
.tran 1u 50u uic
.option eps=1e-4 hmin=5n
.end
```

Simulation Results

Figure C-6. Example#5—Simulation Results



Example#6—Charge Control in MOS 4 and 6

This example deals with the transient analysis of a MOS circuit containing both MOS level 4 and 6 models, illustrating the differences in model performance. The complete netlist can be found in the file *niv4_6.cir*.

Complete Netlist

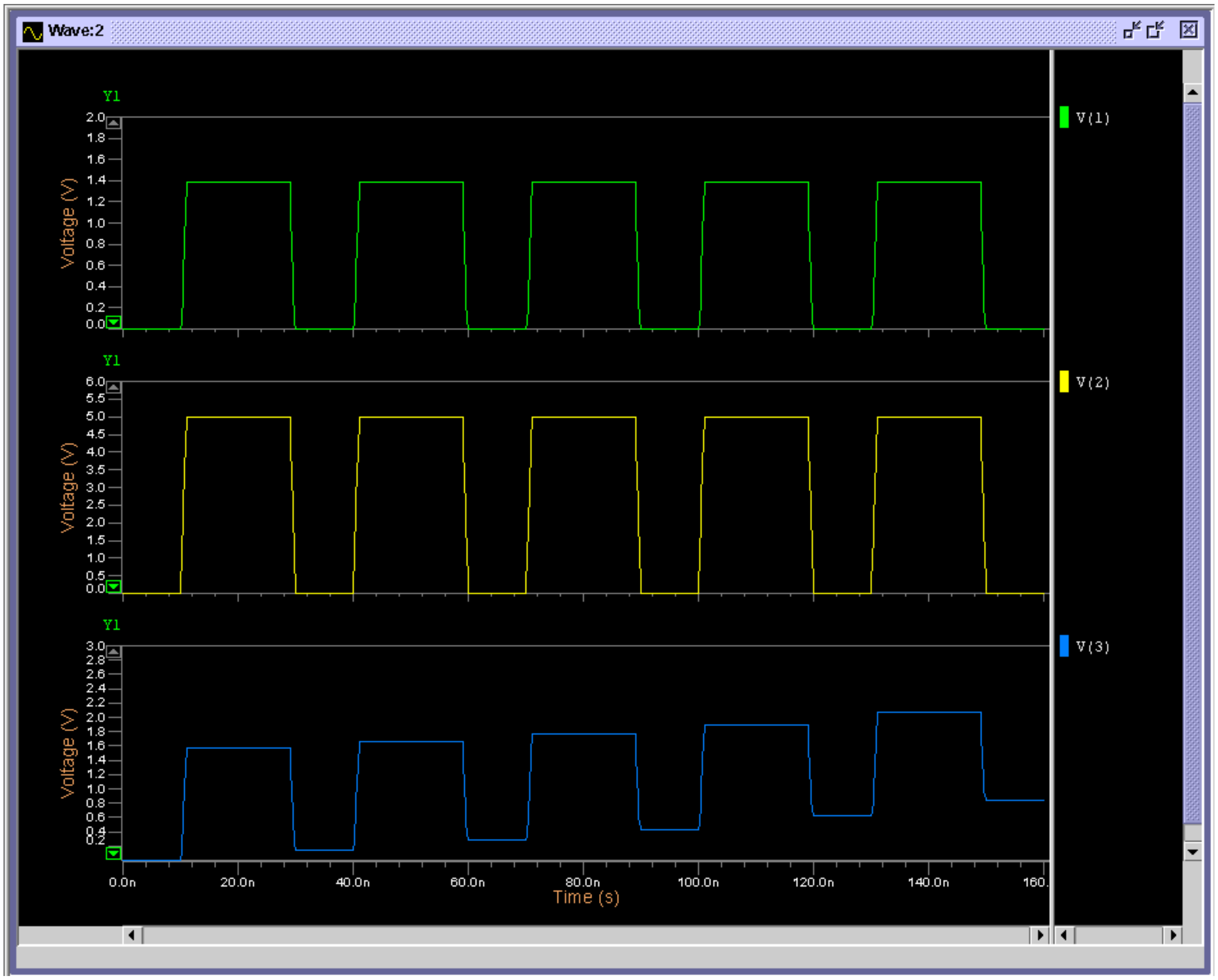
```
niv4_6
.model m4 nmos level=4 vt0=1.2v eox=25n uo=600
+ nsub=2.0e16 phi=0.6 vmax=2.0e5 kw=2.24u kl=2.24u
+ gw=3.91u gl=0.7u dinf=0.1 kb=0.1 ve=1.0e4
+ ldif=10u cj=0.0001 cjsw=0 dw=0 dl=0.8u rec=0.15u
+ tg=0.06
```

```
.model m2 nmos level=6 vt0=1.1v eox=25n uo=600
+ nsub=2.0e16 phi=0.6 vmax=2.0e5 kw=2.24u kl=2.24u
+ gw=3.91u gl=0.7u dinf=0.1 kb=0.1 ve=1.0e4
+ ldif=10u cj=0.0001 cjsw=0 dw=0 dl=0.8u rec=0.15u
+ tg=0.06
m1  1  2  1  bulk m4 w=10u l=3u
c1  1  0  0.05pf
m2  3  2  3  bulk m2 w=10u l=3u
c2  3  0  0.05pf
vb  bulk 0  0
vin 2  0  pulse (0  5  10n  1n  1n  18n  30n)
.tran  1ns  160ns  uic
.print tran v(1) v(2) v(3)
.plot  tran v(1) (0, 2)
.plot  tran v(2) (0, 6)
.plot  tran v(3) (0, 3)
.option eps=1.0e-6
.end
```

In the simulation below, V(1) is the result of the charge control model and V(3) is the result of the capacitive model. For V(3) the error on the charge is accumulated and error is increasing at each period. However, for V(1) there is no error on the charges.

Simulation Results

Figure C-7. Example#6—Simulation Results



Example#7—Active RC Band Pass Filter

This example deals with the AC analysis of an active RC band pass filter circuit containing a UA741 operational amplifier. The complete netlist can be found in the file *ua741.cir*.

Complete Netlist

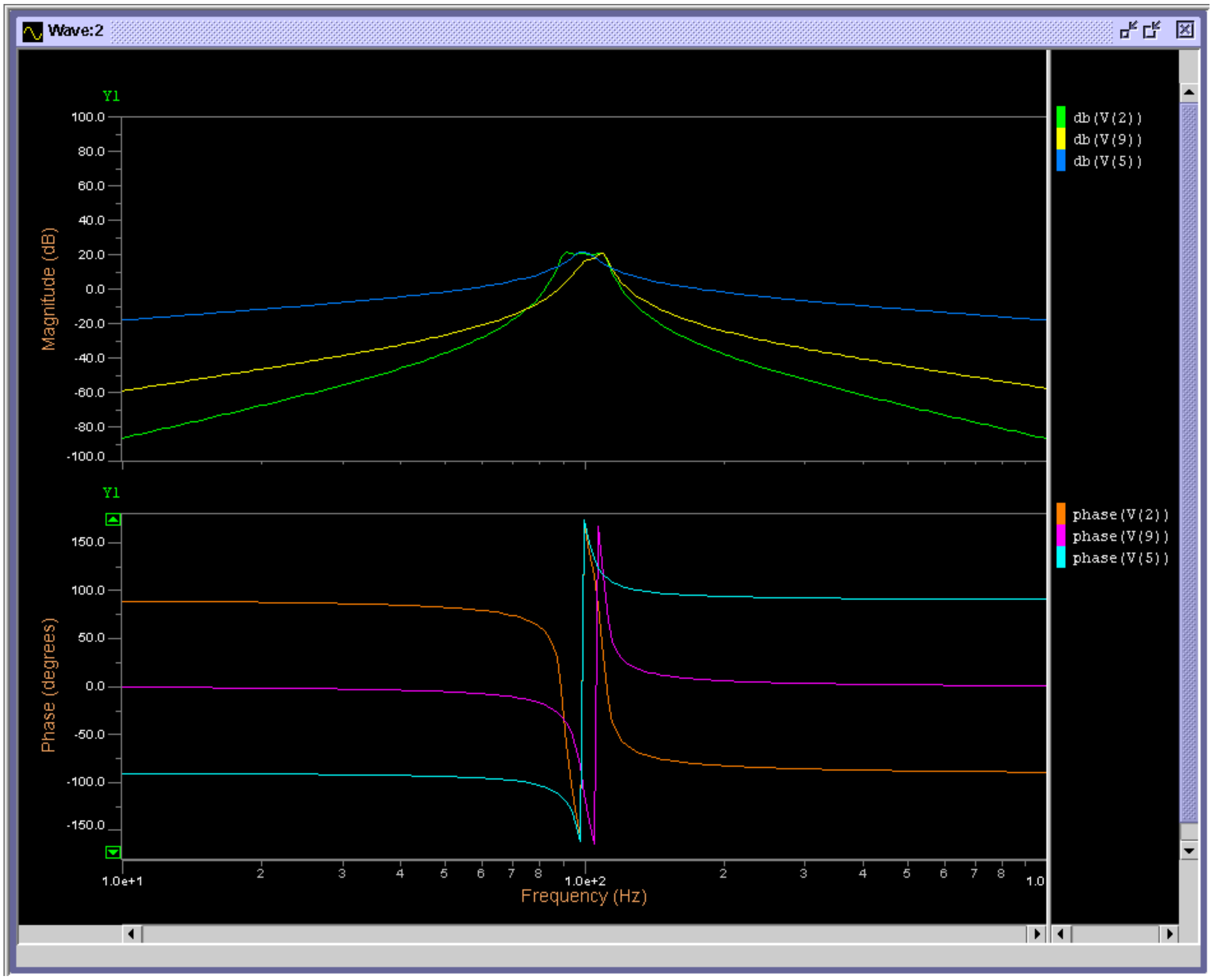
```
ua741
* .MODEL definitions
.model npn npn bf=160 rb=100 cjs=2p tf=0.3n
+ tr=6n cje=3p cjc=2p vaf=100
.model npq npn bf=160 rb=100 cjs=2p tf=0.3n
+ tr=6n cje=3p cjc=2p vaf=100 is=2p
.model pnp pnp bf=20 rb=20 tf=1n tr=20n
+ cje=6p cjc=4p vaf=100
```

```
.model pnq pnp bf=20 rb=20 tf=1n tr=20n
+ cje=6p cjc=4p vaf=100 is=2p
*subcircuit ua741
.subckt ua741 2 3 6 4 7
r1 1 4 1k
r2 15 4 50k
r3 5 4 1k
r4 17 4 5k
r5 18 16 39k
r6 22 23 4.5k
r7 20 23 7.5k
r8 21 4 50k
r9 19 4 50
r10 24 6 25
r11 6  25  50
cx 22 14 30p
q1 9 3 10 npn
q2 12 13 10 pnp
q3 9 2 11 npn
q4 14 13 11 pnp
q5 12 15 1 npn
q6 14 15 5 npn
q7 7 12 15 npn
q8 9 9 7 pnp
q9 13 9 7 pnp
q10 13 16 17 npn
q11 16 16 4 npn
q12 18 18 7 pnp
q13 14 19 4 npn
q14 20 14 21 npn
q15 22 18 7 pnp
q16 22 23 20 npn
q17 20 21 19 npn
q18 22 24 6 npn
q19 7 22 24 npq
q20 4 20 25 pnq
q21 6 6 23 npn
.ends
r1 1 3 12.952k
r22 3 0 846.01
r2 4 5 322.2k
c1 3 5 100n
c2 3 4 100n
x1 4 0 5 40 70 ua741
r3 5 6 150k
r4 6 7 293.8k
r5 7 8 15k
r6 8 9 15k
r7 9 10 15k
r8 11 6 14.05k
c3 6 7 100n
c4 10 11 100n
```

```
x2 6 0 7 40 70 ua741
x3 8 0 9 40 70 ua741
x4 10 0 11 40 70 ua741
r9 9 12 47k
r10 12 13 365.6k
r11 13 14 15k
r12 14 2 15k
r13 2 15 15k
r14 16 12 20.71k
c5 12 13 100n
c6 15 16 100n
x5 12 0 13 40 70 ua741
x6 14 0 2 40 70 ua741
x7 15 0 16 40 70 ua741
vee  40 0 dc -15
vcc  70 0 dc 15
v0   1 0 ac 1
.ac dec 100 10 1000
.plot ac vdb(2) vdb(9) vdb(5) (-100, 100)
.plot ac vp(2)  vp(9)  vp(5)  (180, -180)
.end
```

Simulation Results

Figure C-8. Example#7—Simulation Results



Example#8—2nd Order Delta Sigma Modulator

This example deals with the transient analysis of a second order delta sigma modulator circuit. The complete netlist can be found in the file *integrator.cir*.

Complete Netlist

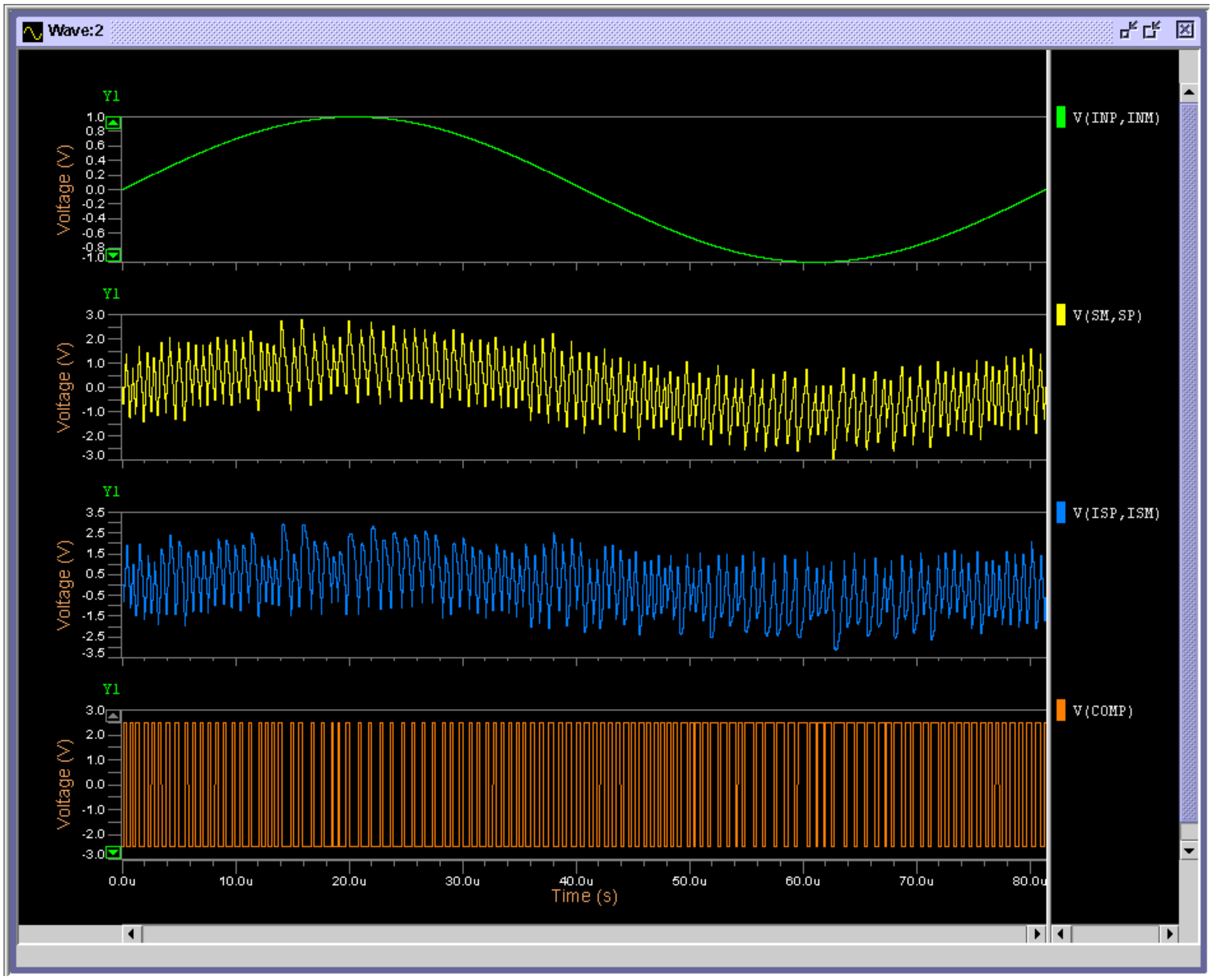
```
integrator
.model switch nsw CREC = 0.0 ron = 100
**** integrator 1 *******
s1  phi   inp 1  switch
s2  phib  1   0  switch
s1  1     2   2p
s3  phi   2   0  switch
s4  phib  2   ep switch
```

```
s5  phi   inm 3  switch
s6  phib  3   0  switch
c2  3     4   2p
s7  phi   4   0  switch
s8  phib  4   em switch
s10 phi   ref 5  switch
s11 phib  ref 6  switch
s12 phib  5   0  switch
s13 phi   6   0  switch
c3  5     7   2p
c4  6     8   2p
s14 phi   7   0  switch
s15 phi   8   0  switch
s16 phib  7   9  switch
s17 phib  8   10 switch
s18 comp  9   em switch
s19 compb 9   ep switch
s20 comp  10  ep switch
s21 compb 10  em switch
c5 ep     sm  6p
c6 em     sp  6p
eopa1 sm sp ep em -1meg
**** integrator 2 ***************
si1 phi sm i1 switch
si2 phib i1 0 switch
ci1 i1 i2 2p
si3 phib i2 0 switch
si4 phi i2 iep switch
si5 phi sp i3 switch
si6 phib i3 0 switch
ci2 i3 i4 2p
si7 phib i4 0 switch
si8 phi i4 iem switch
si10 phi ref i5 switch
si11 phib ref i6 switch
si12 phib i5 0 switch
si13 phi i6 0 switch
ci3 i5 i7 2p
ci4 i6 i8 2p
si14 phib i7 0 switch
si15 phib i8 0 switch
si16 phi i7 i9 switch
si17 phi i8 i10 switch
si18 comp i9 iem switch
si19 compb i9 iep switch
si20 comp i10 iep switch
si21 compb i10 iem switch
ci5 iep ism 6p
ci6 iem isp 6p
eopa2 ism isp iep iem -1meg
vref ref 0 -1
***** comparator *******
compdiff ism isp voutp voutn vhi=2.5 vlo=-2.5 tcom=0.0n
+ tpd=0.0n
```

```
***** latch **************
sl1 phi voutn memn switch
sl2 phi voutp memp switch
nand1 memp qu qubar vhi=2.5 vlo=-2.5 tpd=0.0n
+ vthi=0.1 vtlo=-0.1
nand2 memn qubar qu vhi=2.5 vlo=-2.5 tpd=0.0n
+ vthi=0.1 vtlo=-0.1
***** delay **************
delayn qubar compb 81.3802n
delayp qu comp 81.3802n
***** voltage source input ************
vin1 inp 0 sin(0 0.5 12.288k 0 0)
vin2 inm 0 sin(0 -0.5 12.288k 0 0)
.chrent phib 0 -5 41.6901n -5 42.6901n 5 79.3802n 5
+ 80.3802n -5 81.3802n -5 P
.chrent phi 0 5 40.6901n 5 41.6901n -5
+ 80.3802n -5 81.3802n 5 P
.tran 81.3802n 81.3802us
.option eps=1.0e-9 step=5.0862625n be
.plot tran v(inp,inm) (-1,1)
.plot tran v(sm,sp) (-3.0,3.0)
.plot tran v(isp,ism) (-3.5,3.5)
.plot tran v(comp) (-3.0,3.0)
.end
```

Simulation Results

Figure C-9. Example#8—Simulation Results



Example#9—Operational Amplifier

This example deals with the transient analysis of an operational amplifier circuit. In addition, the slew rate and circuit settling time with a 5% error band are calculated and written to the ASCII output file. The complete netlist can be found in the file *extract.cir*.

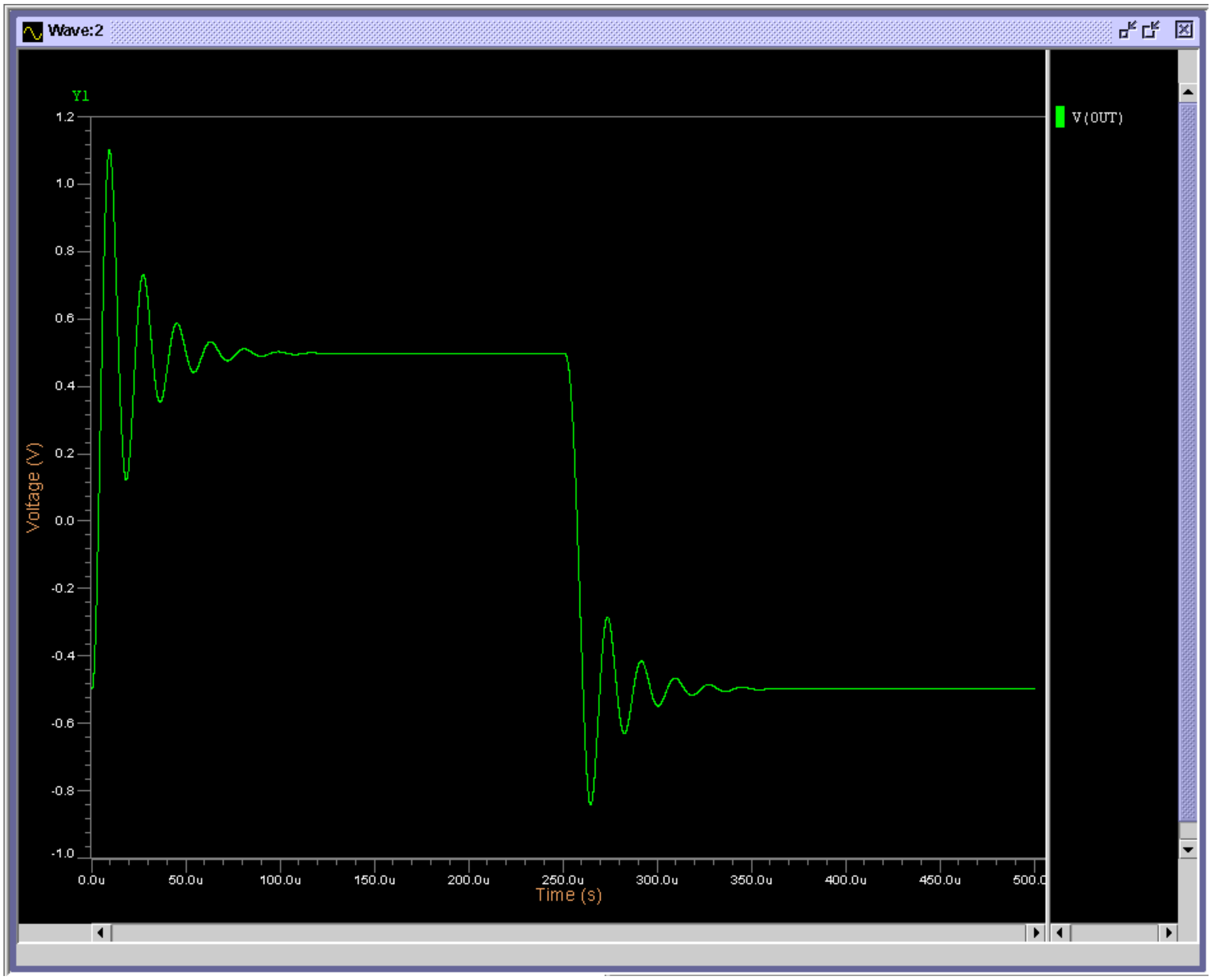
Complete Netlist

```
OPERATIONAL AMPLIFIER
.model bu opa level=2 sl=1e6
c1 out 0 10u
opa1 nonin in out 0 bu
rf in out 2.5k
ri 0 in 0.625k
v1 nonin 0
+ pwl (0.0 -0.1 0.1u 0.1 250u 0.1 260u -0.1 500u -0.1)
```

```
.tran 100u 500u
.option vntol=1e-8 reltol=1e-8 hmax=1u eps=1e-8
.plot tran v(out)
.defmac sett(x,y,tix,limit)=
+ (xycond(x,(y > (yval(y,tix)*(1.0+limit))) ||
+ (y < (yval(y,tix)*(1.0-limit))),tix,0.0))
.defmac slewrate(a)=
+ ((slope(a,yval(a,0.0)+(max(a)-yval(a,0.0))/3.0)+
+   slope(a,yval(a,0.0)+(max(a)-yval(a,0.0))*2.0/3.0))/2.0)
.extract $sett(xaxis,v(out),200e-6,0.05)
.extract $slewrate(v(out))
.end
```

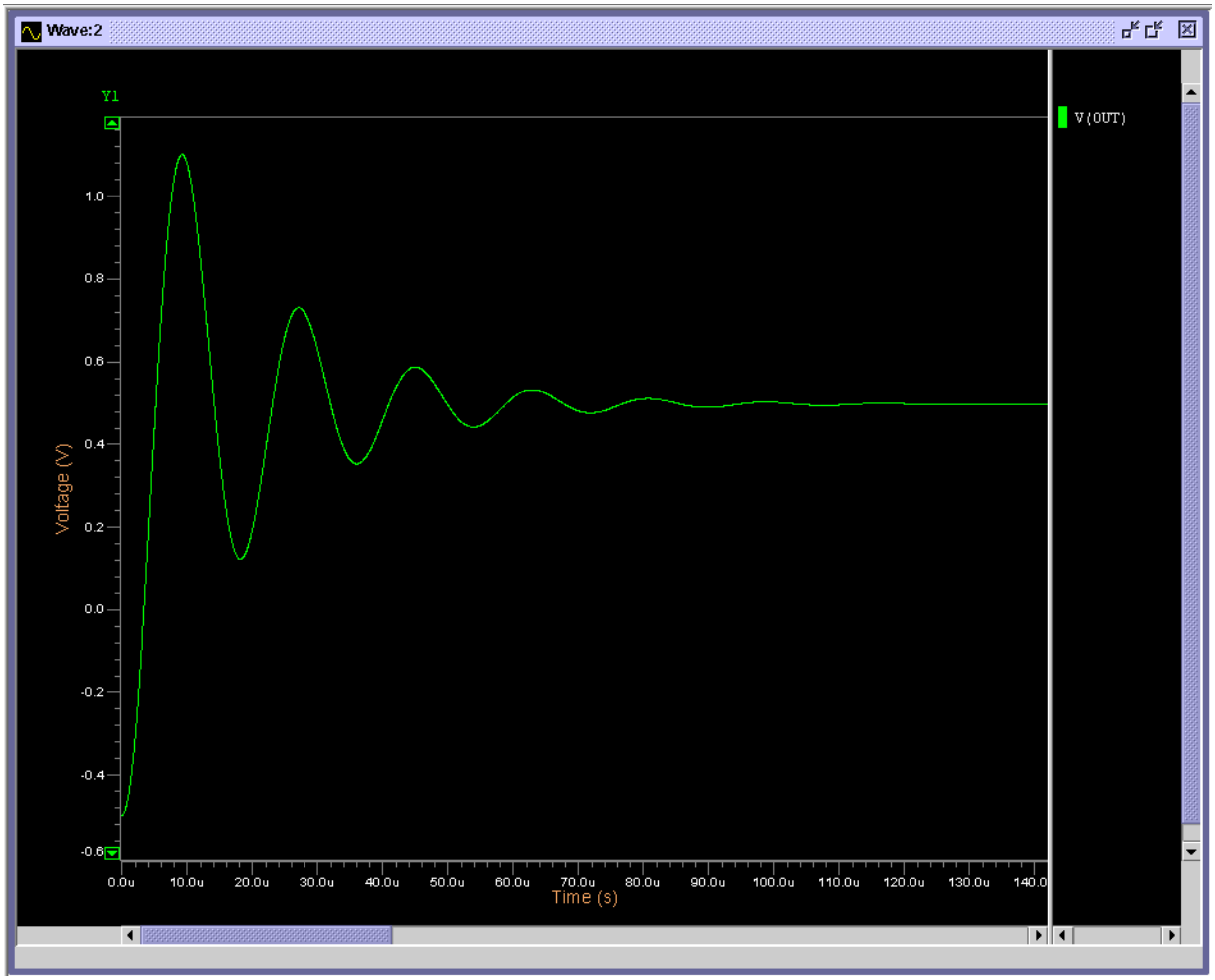
Simulation Results—1

Figure C-10. Example#9—Simulation Results—1



Simulation Results—2 (Zoom)

Figure C-11. Example#9—Simulation Results—2 (Zoom)



Appendix D
Eldo Interactive Mode

Introduction

Eldo Interactive is a way of invoking Eldo and sending commands to it interactively instead of sending the commands in the netlist. Some other AMS tools, such as ICanalyst, make use of Eldo in the interactive mode, although this is all transparent to the user. Eldo Interactive Mode is not supported by Questa ADMS.

To invoke Eldo in the interactive mode, type:

```
eldo cir_file_name -inter
```

where `cir_file_name` is the name of the *.cir* control file to be simulated. Default extension is *.cir*.

When working in Eldo interactive mode, a prompt will appear:

```
eldo>
```

There is some help information available: you can type `help` at the `eldo>` prompt and you will see the list of available commands; type `help <command_name>` for more information about a particular command, or type `help all` to obtain the complete help listing.

When working in Eldo interactive mode, you can type your command after the `eldo>` prompt. For continuation lines, you must type the backslash character (\) at the end of a line.

Commands can be read from a file specified via either of the following:

- `.eil file_name` in the input file
- `-eil file_name` at invocation of the executable

The `eldo>` prompt appears before the first simulation, unless:

- `option DOSIMCIR` is specified
- `-DOSIMCIR` used at invocation of the executable

To control simulation, use the following commands:

```
QUIT
LOAD
RUN
GO
```

```
CONT
NEXT
```

To check netlist elements, use the following commands:

```
DISPLAY
LIST
```

To handle simulation information, use the following commands:

```
STATUS
PRINT
DELETE
RESET
OPTRESET
```

To handle breakpoints, use the following commands:

```
STOP IF (expression)
STATUS BREAK n
DELETE <index>|ALL
```

To alter simulation conditions, use the following Eldo/SPICE commands:

```
.TRAN
.DC
.AC
.OP
.STEP
.TEMP
.ALTER
.NOISETRAN
```

To change stimuli, re-instantiate the device with new values:

```
Vxx n1 n2 <new_stimuli>
```

To change element/model parameters, or parameters, use:

```
SET
```

Use the command, **eilout** file_name, to redirect the outputs generated by specific Eldo interactive commands to the specified file. Errors are still displayed on the terminal window. Use the command, **eilout stdout**, to switch back to the default mode, where information is sent to the terminal window.

To Read Information

STATUS [BREAK]	shows breakpoints
STATUS ANAL	shows status of the current simulation
STATUS AC	shows the **AC** simulation command and stimuli

STATUS TRAN	shows the **TRAN** simulation command
STATUS DC	shows the **DC** simulation command
STATUS MC	shows the **MC** simulation command
STATUS PZ	shows the **PZ** simulation command
STATUS EXTRACT	shows the expressions to extract
STATUS PLOT	shows the **.PLOT** commands typed
STATUS PROBE	shows the **.PROBE** commands typed
STATUS PRINT	shows the **.PRINT** commands typed
STATUS PARAM	shows the parameters (**.PARAM**)
STATUS INPUT	shows the input stimuli
STATUS SAVE	shows the **.SAVE** commands issued
STATUS RESTART	shows the **.RESTART** commands issued
STATUS MCMOD	shows the MC specifications on models
STATUS MCPARAM	shows the MC specifications on parameters
STATUS STAT	shows the statistics at runtime
STATUS OPTION	shows all the options set inside Eldo

PRINT <expression>

The **PRINT** command can be issued to display:

- Voltage or current (like plot command)
- Values from the extract-command language
- **EXTRACT** index
  where index is the key returned by **STATUS EXTRACT**
- Values of Device/Models/Parameters; syntax is:

```
PRINT E (<device_name>[,W/L/AD/AS/PD/PS/NRD/NRS/AREA])
PRINT M (<model_name>,<parameter_name>)
PRINT P (<param_name>)
```

- **OPTION <name>**

Examples:

```
PRINT V(S)
PRINT V(S) + V(SS)
```

```
PRINT TPD(E,S)
```

Please note that the extract information issued through a **PRINT** command work only if the corresponding **.EXTRACT** command had been specified before running the simulation, or if Eldo is able to extract the information from the binary output file (**.PLOT** available, and general-purpose extraction language used).

DISPLAY E string[*]

LIST string*	List all elements whose name begins with *string*
LSMOD string*	List all models whose name begins with *string*
LSMODEV string*	List all elements which have *string** as model
LSSUB string*	List all subcircuits whose name begins with *string*
LSSUBDEV string*	List all instances of the subcircuit whose name begins with *string*

.LPTOP

This command requests Eldo to dump **.PARAM** names which are top parameters and which are not coming from a **.LIB** library file. This is useful when debugging circuits.

.LDEVNODE

Returns the list of devices connected to a node.

.LDEVNODE <node>

.LNODEDEV

Returns the name of the pins a device is connected to. A device name must be specfified as an argument. For example:

```
M1 A B C D NMOS w=10u l = 3u
```

eldo> .lnodedev M1

will return:

```
M1.1 A
M1.2 B
M1.3 C
M1.4 D
```

.LSXINST

Returns the list of X instances matching a condition (with wildcards).

eldo> .lsxinst <filter>

TRACEI

```
tracei <node_name> [thresehold_value]
```

Returns the current contributions on the specified node. For example, under the eldo> prompt, you can specify:

eldo> tracei s 1.0e-8
IX(M4.1) = -2.1725E-05
IX(M9.1) = 2.1725E-05

The two contributions are not dumped because they fall below the threshold 1.0e-08. This **`tracei`** command returns current contributions on the specified node (s in this example): 1.0e-8 is a threshold, optional.

To Reset Several Features

`RESET [PRINT|PLOT|PROBE|IC|NODESET|GUESS|EXTRACT]`
Used to remove a set of commands.

`RESET FILES`
Rewind output files *.cou* and *.chi*.

Options

DC Control Options

```
GMIN
NMAXSIZE
ITL1
GRAMP
NETSIZE
VMIN
VMAX
```

Accuracy Control Options

```
ITOL
EPS
VNTOL
RELTOL
RELERR
PIVREL
```

```
PIVTOL
ABSTOL
FLXTOL
MAXORD
```

Time-step Control Options

```
ZOOMTIME
STEP
STARTSMP
FREQSMP
OUT_RESOL
TRTOL
HMIN
ITL3
ITL4
FT
DCLOG
LVLTIM
LVLCNV
DVDT
RELVAR
ABSVAR
SAMPLE
HMAX
SPICDC
NOSPICDC
```

Options which can be set, but not reset:

```
SPIOUT
NEWTON
OSR
TRAP
GEAR
BE
PROBEOP
NOLAT
NWLAT
ANALOG
BBDEBUG
NOSIZECHK
QTRUNC
UNBOUND
LCAPOP
```

Change Features

Some Eldo commands are available from interactive mode; they are:

```
.TRAN .AC .DC .OP .STEP .TEMP
.PLOT .PRINT .PROBE .EXTRACT .OPTION
.MEAS .FOUR .OPTFOUR
.NODESET .IC .GUESS
```

```
.SAVE .RESTART .USE .LIB
```

It is possible to change the stimuli in interactive mode. The complete line must be re-entered with new stimuli.

Example:

```
V1 1 2 PWL (0 0 20n 0 30n 5)
```

The program checks the component name, and applies the new stimuli.

BUS

```
BUS <name> <type> <list_of_signals>
```

Define a bus. The **BUS** command defines a bus with several different nodes by grouping them together.

<name> Name of the new bus

<type> Ignored (kept for Lsim compatibility)

<list_of_signals>
List of nodes composing the bus (msb...lsb).

Example:

```
bus foo x A[0] A[1] A[2] A[3] clk reset
bus foo x A[0:3] clk reset
bus ADD  x A0 A1 A2 A3 A4 A5
```

[.]CHMOD

```
CHMOD <model1> <model2>
```

Replace the model `<model1>` by the model `<model2>` for all the devices using `<model1>` (can be used for M, Q, D, J models).

DELETE

```
DELETE BREAK index
DELETE BREAK ALL
DELETE <cmd> index
```

`DELETE` is used to remove breakpoints used to stop Eldo at run time. The third syntax is used to remove the command corresponding to the index returned by the command `STATUS <cmd>`.

DISABLE

```
DISABLE BREAK index
DISABLE BREAK ALL
```

DISABLE is used to remove breakpoints used to stop Eldo at run time. Breakpoints can be enabled back using command ENABLE.

ENABLE

```
ENABLE BREAK index
ENABLE BREAK ALL
```

ENABLE is used to re-activate breakpoints used to stop Eldo at run time, whenever these breakpoints have been disabled by DISABLE.

FORCE

```
FORCE <node_name>=<value>
```

Force the node node_name to value after **RISE_TIME** or **FALL_TIME** depending on current value (see further information below). Used to impose a voltage on a node. If multiple FORCE are applied on the same node, the last command is used. If an input signal was applied on the node it will be ignored. The voltage imposed by FORCE can be removed using RELEASE.

HIGH

```
HIGH <node_name>
```

Force the node node_name to **HIGHVOLTAGE** after **RISE_TIME** (see further information below).

LOW

```
LOW <node_name>
```

Force the node node_name to **LOWVOLTAGE** after **FALL_TIME** (see further information below).

The previous three commands contain the parameters, **HIGHVOLTAGE**, **LOWVOLTAGE**, **RISE_TIME** and **FALL_TIME**. These parameters can be set using the **.OPTION** command. For more information, see "Miscellaneous Simulation Control Options" on page 974.

PWL

```
PWL <node_name> t1 v1 [t2 v2] ...
```

Force a PWL on the specified node **<node_name>** during transient simulation. *tn* are times relative to the current time. *vn* are the source values at *tn*.

RELEASE

```
RELEASE <node_name> ...
```

Release force on node. If a signal is present on node **<node_name>**, and if no **FORCE** command had been applied on the node, then the signal is disabled (node will be computed by Eldo as any other node).

SET

```
SET E (<element_name> [,W/L/AD/AS/PD/PS/NRD/NRS/]) [=] <value>
SET M (<model_name>,<parameter_name>) [=] <value>
SET P (<parameter_name>) [=] <value>
```

The SET command is used to change device geometries or models parameters. The parameter_name can be specified as a string parameter.

SETBUS

```
SETBUS <hierarchical_name>[[<msb>:<lsb>]] <type> <value>
```

Set a value on a bus. The **SETBUS** command sets a value on the specified bus. Values may be specified as binary, octal, hexadecimal, or decimal. The bus bit accepts the standard digital values 1, 0, X and Z.

The command automatically sets all signals in a bus if the subscripted values are omitted. The default bit order is the European style $D_0...D_{N-1}$. Therefore, for a hierarchical base name that defines an 8-bit bus, the command:

```
setbus Xcircuit.port[0:7] b00001111
```

is equivalent to:

```
setbus Xcircuit.port b00001111
```

To use the American style $D_{N-1}...D_0$ bit order, you must specify the range of bits explicitly:

```
setbus Xcircuit.port[7:0] b00001111
```

The **SETBUS** command ignores the most significant portion of values that are wider than the specified bus. Therefore, if you set a value of hexadecimal 8F to a 12 bit bus, the bus receives the value 0F.

Note

This **`SETBUS`** syntax is different to the Eldo **`.SETBUS`** syntax used in the netlist *.cir* file. See ".SETBUS" on page 870 and ".SIGBUS" on page 880. However, if you want to use the netlist syntax in interactive mode, specify the Eldo interactive command:

```
eldo>LSIM OFF
```

TRANSITION

```
TRANSITION &lt;node_name&gt; TT VALUE [DELAY]
```

TT	Transition Time
VALUE	Value of the signal after the transition
DELAY	Time delay before the transition starts

Overwrite the signal which was on node **`<node_name>`** by a PWL:

```
Vxx <node_name> 0 PWL (<current_time> <current_value>
<current_time + DELAY> <current_value>  <TT + DELAY> <VALUE>)
```

To Control Execution

LOAD <filename>

This command stops the current simulation, and causes Eldo to load the file <*filename*>. Extension *.cir* is assumed.

SAVESIM <filename>

Eldo creates three files:

1. *filename.eil*
   This file contains all the commands typed since the loading of the circuit file; it is possible to re-execute this set of functions by:

   ```
   <eldo> -i circuit -eil filename.eil
   ```

2. *filename.chi*
   This file contains the ASCII output

3. *filename.cou*
   This file is the binary output file readable by graphic-postprocessors

Note

A **`.pz`** file might be created if a **`.pz`** card exists.

The files *circuit.cou* and *circuit.chi* are then reset.

STOP IF <condition>

Set breakpoints in interactive mode. **`<condition>`** can refer to the **`SWEEP`** value, and/or any valid plot command.

The argument, **`soadetected`**, can be specified to specify a breakpoint on a safe operating area (SOA).

Examples

```
STOP IF (SWEEP > 1n)
STOP IF ((SWEEP > 10n) && (V(S) > 2.5))

stop if(soadetected(mysoa))
```

Eldo will stop the first time the **`.SOA`** labeled *mysoa* is violated. Wildcards are allowed in soadetected.

STOP SIMU

Stop the current simulation; Eldo is then ready to run a new simulation.

RUN

Restart the simulation.

`RUN FOR <value>`	Run until the sweep value has changed of **`<value>`**
`RUN UNTIL <value>`	Run until the sweep value reaches **`<value>`**

Note

`<value>` can be a parameter, this must be specified on the **`.PARAM`** command in the netlist.

NEXT [SIMU]

`NEXT`	Eldo will stop at the next sweep value.
`NEXT SIMU`	Eldo will stop after the next simulation if **`.TEMP`** or **`.STEP`** command are found.

CONT

`CONT`	Forces Eldo to continue the simulation(s) until breakpoints are encountered, or the requested number of simulation is completed.

QUIT

`QUIT`	Quit the application.
`QUIT SIMU`	Stop current analysis.

CONNECT XELGA

Used to connect Eldo to Xelga in Eldo standalone mode.

VIEW plot_name
UNVIEW plot_name

Used for adding or removing signals whenever Xelga is connected to Eldo, running in interactive mode.

Appendix E
Eldo Utilities

Utility to Convert .chi to .cir

A standalone utility **`chi2cir`** converts an Eldo ASCII output (*.chi*) file into a netlist (*.cir*) file. The resulting netlist file is independent of any libraries (no **`.LIB`**/**`.INCLUDE`** required), and all information required for another simulation is inside the netlist file (providing that there were no commands such as **`.NOTRC`** preventing the writing of the circuit description in the *.chi* file).

This could be useful to exchange testcases between user's avoiding library dependence and would also simplify the creation of different tests on the same circuit.

Once the circuit has been run once (to generate the *.chi* file), run this **`chi2cir`** utility to generate a testcase from this *.chi* output in order to modify some parameters/options to re-run other simulations.

Usage

```
chi2cir [chifile cirfile]
```

- `chifile`

  Input filename, the Eldo ASCII output (*.chi*) file.

- `cirfile`

  Output filename for the resulting netlist (*.cir*) file.

If no options are specified, the utility will prompt you for filenames.

Example

```
chi2cir trigger.chi my_trigger1.cir
```

Once the original *trigger.cir* circuit had been run once (to generate the *.chi* file), running the **`chi2cir`** utility on the *trigger.chi* ASCII output file generates a testcase file *my_trigger1.cir*. Some parameters/options can then be modified in this testcase to re-run other simulations.

Notes

- The chi2cir utility does not handle references to Verilog-A model files or S-parameters files.
- No provision is made for **`.ALTER`** statements for multiple simulation runs, only the first simulation is taken into account.

Utility to Convert VCD to Test Vectors

A standalone utility **`vcd2tv`** can be used to convert VCD (Value Change Dump) stimuli into test vectors. It is a simple format converter, and can be used independently for any simulator. Output is in the test vector format. This utility can also be used to convert EVCD (extended VCD) files. Note when converting from EVCD format: inout statements are converted to input statements.

The ADiT fast SPICE simulator supports VCD directly because it internally converts it to test vector format, transparently using the **vcd2tv** converter. Eldo is not able to read VCD, which is why this utility is useful. It can be used for any simulator which can read test vectors but not VCD.

After converting a VCD file, use the generated test vector file by specifying the .TVINCLUDE command.

Usage

```
vcd2tv -i vcd_input_file [-io io_input_file] [-o tv_output_file]
[-hier] [-end_time conversion_end_time]
```

- `-i vcd_input_file`

  Input filename. Name of the VCD file to convert.

- `-io output_file`

  Input/output filename. Name of the file containing the I/O definition (input or output).

- `-o tv_output_file`

  Output filename. Name of the test vector file (*.tv*) to generate.

- `-hier`

  When specified hierarchical node names are also converted. If not specified, only the top level is converted.

- `-end_time conversion_end_time`

  Conversion time limit (if not specified conversion is done until the end of the VCD file).

Utility to Convert a Waveform Database

A standalone utility **`ffcv`** can be used to convert a waveform database from one format to one, or several, other formats.

Usage

```
ffcv input_database output_format_modifier [output_database]
     [-cvt_start xstart] [-cvt_stop xstop] [-cvt_tsample xvalue]
```

- `input_database`

  Input filename with known extension. Name of the waveform database file to convert.

- `output_format_modifier`

  Specification of the output format to generate. Can be one of the following:

Table E-1. ffcv Output Format Modifier

output_format_modifier	Output Format
-cou	Binary COU file
-cou_ascii	Raw ASCII representation of a COU file
-comment	Commented ASCII representation of a COU file
-compat	Binary compatible file
-compat_ascii	ASCII compatible file
-csdf	CSDF file
-fsdb	FSDB file
-wdb	Joint Waveform Database file (Eldo default)
-stimuli	SPICE representation of voltage waveforms
-tabular	Tabulated representation of waveforms
-wdf	WDF file

- `output_database`

  Output filename. Name of the waveform database file to write converted format to.

- `-cvt_start xstart`

  Conversion start point (time or frequency) in database.

- `-cvt_stop xstop`

  Conversion end point (time or frequency) in database.

- `-cvt_tsample xvalue`

  Time or frequency interval to sample output data.

Appendix F
Eldo Encryption

Eldo Encryption

encrypt_eldo is the tool to encrypt the libraries containing **.SUBCKT** definitions, **.MODEL/.PARAM** cards, and **.PROTECT**/**.UNPROTECT** blocks. It uses DES encryption with an internal 56-bit key. The file will be automatically decrypted by Eldo at run time, but none of the encrypted information will be displayed in the ASCII output file (*.chi*). This guarantees the confidentiality of the data.

See the .PROTECT and .UNPROTECT command descriptions for further information.

Caution

Apart from **.PROTECT**/**.UNPROTECT** blocks, **.MODEL**, **.PARAM**, and **.SUBCKT** commands will be encrypted.

Usage

```
encrypt_eldo -i input_file [-o output_file] [-compat] [v]
```

- `-i input_file`

  Input filename. The file can contain model cards, parameter cards, subcircuit definitions, and protected blocks for encryption.

- `-o output_file`

  Output filename. If this is not specified, the filename will be *input_file.crypt*.

- `-compat`

  Simulator compatibility argument. When specified only **.PROTECT**/**.UNPROTECT** blocks will be encrypted.

- `-v`

  Returns the software version number. No encryption is performed.

Example

1. Unencrypted netlist (*diode.lib*):

```
.model DNPPSJU
+ d level=8 diolev=9 tr=27             tnom=27
+ vr=0               cjbr=0.00072727   cjsr=8.649e-12
+ cjgr=3.946e-10     jsdbr=3.5987e-07  jsdsr=3.4372e-12
```

```
+ jsdgr=6.9543e-14   jsgbr=6.0153e-06     jsgsr=1.0506e-09
+ jsggr=6.4326e-10   vdbr=0.55572         vdsr=0.49482
+ vdgr=0.79381       pb=0.44466           ps=0.99
+ pg=0.43889         nbj=1                nsj=1.2
+ ngj=1

.subckt diode A B

.model my_diode D diolev=9 tr=27          tnom=27
+ vr=0                 cjbr=0.00072727   cjsr=8.649e-12
+ cjgr=3.946e-10       jsdbr=3.5987e-07  jsdsr=3.4372e-12

R1 A A1 1k
D1 A1 B1 my_diode
R2 B1 B 1k
.ends
```

2. Command for encrypting this netlist with the Eldo license:

```
encrypt_eldo -i diode.lib -o diode_crypt.lib
```

3. Output file generated (*diode_crypt.lib*):

```
%.model DNPPSJU
%6A41CAC8CE3D50CB0B85F0126F6D1CCF23E5CA9C3BD0E7F21716251CB19DA7
%36FBBF69EF23E826A0478CA39376DCC8C4B7DBCD4A5A61CD26999C7F74FA2C
%75D9133AD4A91885751DDCE235FB1D944B96A4491CE3EDC5247588A7697FA1
%F912ED3488C832AB598321B58D3F25D176D794F6376924596B7948A4068996
%35363770AA370F3C10331A8FC0B5822FFDA963774FFFC74F1FFA3021EA693F
%D88A1A3E8DDBC62012F317FE3BF379A999B6638BFE8A7A97E6B46C6E1E2B60
%A2A9362162265083A1A898E5CF8EF5A5F6FF420A52C2E23FB9364E2BC13F29
%6F79757FF86BB080DCBAA2B4FC7EBF863B3B0C7C896BAF6525DF00901CD9E1
%6FC7E1BD4E691E6B006451AAE3302D64AD912114D5AC4F3D436BB1AAE5582F
%1A

.subckt diode A B
%A40ACD98849922AC04809D607E2424D4255F6765FF779A2EF49C24DC0
%483152A2024CA2B0C2E975EE8E41C1DCD2098C152082EA626C990AE64
%7B7F38B8FFEA9868E28164C55E19139F8378A9FF6BB9946CC7448D744
%127ADEA26B511101C631E3D455880E6E56EFBFEBC481964A7300C9FBF
%5B57245A6CC15D2A72BC833D8437DCF53D85F1B017366485676443756
%2BFFAC6DFEB58B5A43ACD64C30ECA1F7491380ED4F8777FBDF2BF6DA1
%287BD1F4C86D96B25ABA5164851F196324189F42BFC67116235368287
%5
.ends
```

4. Input file (*test.cir*)

```
* example for encryption netlist
.lib diode_crypt.lib
v1 1 0 1
d1 1 2 DNPPSJU
x1 2 0 diode
.op
.end
```

5. Invoking Eldo.

```
eldo test.cir
```

Notes

The netlist is not displayed inside the Eldo output (*.chi*) file, as shown below:

```
.MODEL DNPPSJU   ----> the lines 4 to 13 are not displayed here.

.SUBCKT DIODE A B   -> the lines 16 to 23 are not here.
Warning 902: “DIOLEV in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “TR in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “VR in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “CJBR in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “CJSR in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “CJGR in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “JSDBR in model DIODE.MY_DIODE”: Model parameter ignored.
Warning 902: “JSDSR in model DIODE.MY_DIODE”: Model parameter ignored.
.ENDS


V1 1 0 1
D1 1 2DNPPSJU
X1 2 1 DIODE
```

The encrypted model parameters are not displayed inside the Eldo output *.chi* file.

Protection of Encrypted Libraries

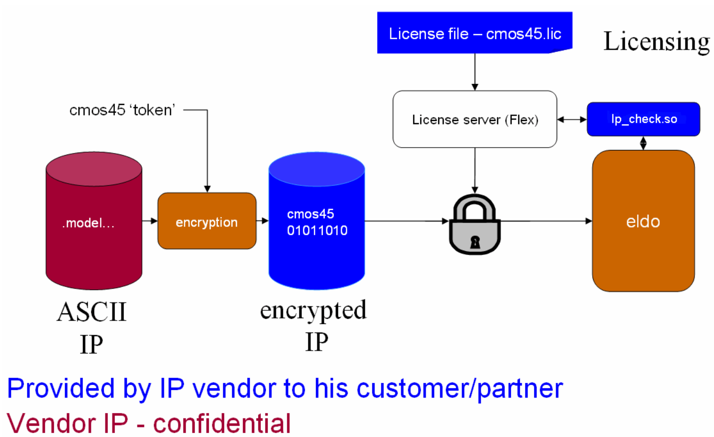
Overview

An improved IP protection system is available, providing an additional level of protection compared to simple encryption available through the encrypt_eldo utility. This system is for IP providers with respect to their customers. The device model libraries provided by IP providers (typically foundries here) often contain process sensitive information such as SPICE device models, equations, technology parameters, and so on. These may be encrypted, but there is no easy way to restrict the usage of these encrypted model libraries. In particular, there is no way to restrict the allowed duration of usage. The purpose of the new IP protection system is to allow an IP provider to ship encrypted and licensed model libraries to their customers, independent of Mentor Graphics. The licensing mechanism might be a standard commercial license managing system. Using the system, the IP provider can control the usage of its model libraries in exactly the same way that any commercial software is controlled, using features, license files, expiration dates, and so on. The system is designed in such a way that Mentor Graphics does not have to know the final end-users, nor generate any keys nor licenses. The IP provider creates and provides both the encrypted IP (the model libraries) and an “IP access library”. The IP access library is a dynamic load library (compiled C code). Mentor Graphics does not interfere with the encryption process, nor the license generation, maintenance, renewals, and so on, which are the entire responsibility of the IP provider.

The steps involved in preparing and installing a vendor-specific protected/encrypted library are as follows:

1. The vendor, or IP provider, prepares a single “IP access” library. This is a dynamic load library coded in C.
2. The vendor then prepares and encrypts the Eldo library source for each library the vendor wishes to distribute.
3. The user installs the vendor-specific IP access library and the vendor protected/encrypted libraries.
4. The user then installs the licensing software and keys provided by the IP vendor.

Figure F-1. IP Protection



Coding and Encrypting a Protected Library

As an IP provider wishing to protect a library, you must include these specific commands in your library file (.lib):

```
.IP_protect IP_provider=<NameOfTheIPProvider>
+ IP_access_lib=<NameOfTheAccessLibrary> feature=<NameOfTheFeature>
+ ixl=<ixl_crc> ss5=<ss5_crc>
+ ixl64=<ixl64_crc> ssw=<ssw_crc>
```

The **`.IP_protect`** command must be written inside a **`.PROTECT`**/**`.UNPROTECT`** area of the device model library (**`.LIB`**). The device model library then has to be encrypted using the standard eldo encryption tool, **encrypt_eldo**, which is provided with the AMS distribution. As the IP_protect command is inside a **`.PROTECT`**/**`.UNPROTECT`** area, it is encrypted as well, and therefore cannot be read nor changed by the final user. See the command description for “.PROTECT” on page 850 for further information.

The meaning of the items in the **`.IP_protect`** command are explained below.

- IP_provider

  The name of the IP provider, as it will appear in messages. Use a simple string comprising only letters (A-Z) and numbers (0-9). Do not use blanks.

- IP_access_lib

  The name of the dynamic load library which communicates between Eldo and the actual license manager. Note that *NameOfTheAccessLibrary* must be given without its extension. Eldo will add the proper extension (such as "*.so*") according to the platform. This library will have to be located in a path listed in the LD_LIBRARY_PATH environment variable.

- feature

  The feature name which will be checked-out to grant access to the protected library.

- ixl, ss5, ixl_64, ssw

  These are CRC values of the IP access libraries, as returned by the eldo_checksum.exe utility provided in the AMS distribution. These CRC values must be provided, and are checked by Eldo to verify the integrity of the IP access library. The eldo_checksum.exe utility must be run once per physical platform.

Description of the Loading and Control Process

Whenever an Eldo netlist contains a reference to an encrypted model library containing a **`.IP_protect`** command, Eldo reads the name of the IP access library from the **`.IP_protect`** command, and then dynamically loads that IP access library (NameOfTheAccessLib). This dynamic library is used as an interface between Eldo and the license daemon of the IP provider (or any other licensing manager). Eldo communicates with this dynamic library only to ask whether the loading of the model library is allowed, by calling a simple function. The dynamic library in turn communicates with the license manager of the IP provider. It is the IP provider's responsibility to provide the end-user with the daemon, license file, and so on.

If the access is granted, Eldo decrypts and loads the necessary devices from the .lib file, and runs the simulation. If access is denied, the simulation aborts. Notice that when using an encrypted library, the **`.LIB`** command in the netlist is a regular **`.LIB`** command (that is, whether the device model library is encrypted or not does not change the usage).

Eldo decides if access is granted or denied by calling a function in the IP access library. This function must be coded by the IP provider, and it is in charge of interrogating the provider's license daemon to grant/deny access to the necessary feature. This function is described later.

As it would be too easy to fool this system by replacing the IP access library by another dynamic library which always returns 'ALLOWED', Eldo verifies that the checksum of the loaded library is the same as the one specified in the **`.IP_protect`** command. That means that if a new IP access library is compiled, the **`.IP_protect`** command must be rebuilt, and the model library re-encrypted, as the CRC will have changed. Note that a CRC is platform dependent, and therefore several CRCs must be provided in the **`.IP_protect`** command to accommodate the various targeted platforms (ixl, ss5, and so on).

The standalone executable, **eldo_checksum.exe**, generates the CRC of the library. This is available in the standard AMS distribution, located in the *$MGC_AMS_HOME/$VCO/bin* directory.

For example, run:

$MGC_AMS_HOME/$VCO/bin/eldo_checksum.exe <NameOfTheAccessLibrary>

where *NameOfTheAccessLibrary* is a *.so* file on Solaris or Linux. Specify this name without its extension.

The utility returns a CRC value which must be provided to the **`.IP_protect`** command. It must be run once for each platform.

The NameOfTheIPprovider character string is used to display proper messages to the final user if there is a problem during the license check-out.

Creating the IP Access Library

The required Eldo interface of the IP access library must contain two functions, namely:

```
ty_lib_status * get_license_third_party(char *feature);
```

and

```
ty_lib_status * release_license_third_party(char *feature);
```

The ty_lib_status type and the function prototypes are defined in the *eldo_ipprotect.h* include file, shown below. This is provided in the *$MGC_AMS_HOME/include* folder. The various return values and their meanings are explained in the comments.

Contents of eldo_ipprotect.h Include File

```
typedef enum
{
  /*! The license manager has successfully checked out the feature,
   *  nothing special to do */
  getFeatureOk,

  /*! The license manager failed to check out the feature.
   *  An error message will be printed and the simulation will stops.*/
  getFeatureNOk,

  /*! The license manager has checked out the feature, but a warning
   *  should be printed
   *  (for example: if the license is about to expire soon) */
  getFeatureOkWarn,

  /*! The license manager has released the feature, nothing special to do */
  releaseFeatureOk,

  /*! The license manager failed to release the feature. A warning message
   *  will be printed and the simulation will continue.*/
  releaseFeatureNOk,

  /*! The license manager has released the feature, but a warning
```

```
     *  should be printed */
    releaseFeatureOkWarn
  } en_ipstatus;


  /**
   * This structure is used by eldo to check the result of the license
   * request.
   */

  typedef struct
  {
    /* The status of the request. According to it some actions will be done.*/
    en_ipstatus status;

    /* The message that eldo must print. Once the message is printed, eldo
     * will free the memory allocated for the message.
     */
    char *message;
  } ty_lib_status;


  /**
   * \brief This function is used to check that the feature given in
   *        argument is available. Eldo doesn't manage the possible queue
   *        systems. It is up to the library loaded by eldo to do that job.
   *
   * \param feature is the feature name (as it appears in the license file)
   *  needed to continue to parse the design
   *
   * \returns the status of the license check-out.
   *          If the license check-out failed, eldo will print the attached
   *          message as an error, will free it just after,
   *          and will abort the simulation. It will work the same way
   *          if the status is getFeatureOkWarn but the printed message
   *          will be a warning and it will not abort the simulation.
   */
  ty_lib_status * get_license_thrd_party(char *feature);


  /**
   * \brief That function is used to release the feature given in argument.
   *        Eldo doesn't manage the possible queue systems. It is up to the
   *        library loaded by eldo to do that job.
   *
   * \param feature is the feature name that has been previously checked out
   *  and that must be released.
   *
   * \returns the status of the license checkin.
   *          If the license checkin failed, eldo will print the attached
   *          message as a warning and will free the message just after.
   *          The simulation will not be aborted. It will work the same way
   *          if the status is releaseFeatureOkWarn.
   */

  ty_lib_status * release_license_thrd_party(char *feature);
```

The compilation command for the shared library is as follows:

gcc -I$MGC_AMS_HOME/include (...other include folders...) <NameOfTheCFile.c>
-o <NameOfTheSharedLibrary> -shared

This library does not need to be recompiled with each new version of Eldo **unless** this is explicitly stated in the release notes of a new version.

Installing a Protected Library

For Eldo to be able lo load a protected library, the IP access library (a *.so* file on Solaris or Linux) must be located in a directory named *.../ixl* or *.../ss5*, and so on, depending on its binary origin (Linux, Solaris, and so on). The parent of that directory must be specified in the standard LD_LIBRARY_PATH variable.

For example:

Having received the *foo.so* library (for Linux 32) from your provider IpProvider, you want to put that library under the common shared folder */shared/common/thirdPartyLibs*. You must create a specific folder according to the platform of the *foo.so* library. In this example it is a Linux 32-bit library so put the file under */shared/common/thirdPartyLibs/ixl*.

The path to include in the LD_LIBRARY_PATH variable is */shared/common/thirdPartyLibs*. Eldo will automatically search under the *ixl* folder if it is run on the Linux 32-bit system.

List of Errors and Warnings

The various “%s” appearing in these messages are place-holders for the actual names of the libraries, names, and so on.

```
ERROR 1608: Wrong vendor library, found at:\n+ %s\n+ The CRC of that
library is not the expected one. Please contact your %s office.
```

The checksum made on the found library does not match the one specified in the netlist.

```
ERROR 1609: Unable to find the %s%s dynamic library provided by %s.\n+
Check first your LD_LIBRARY_PATH. It must contain the path to %s/%s%s.\n+
If you don’t have that library, please contact your %s office to get one.
```

Eldo was not able to find the requested library through the folder given in the LD_LIBRARY_PATH variable (with respect to the rule specified above).

```
ERROR 1610: The protected part of the %s design requires a license to be
used. Eldo is unable to get a valid license feature %s\n+ %s”
```

Checkout of the license has failed.

```
ERROR 1611: Shared library %s%s was found but an error occured during its
loading.\n+ Please contact %s office.
```

An unexpected error has occurred (there was a missing function, the simulation was aborted before, the crc has been changed, and so on).

```
ERROR 1612: Wrong vendor library %s%s found. The CRC of that library is
not the expected one. Please contact your %s office.
```

The checksum made on the found library does not match the one specified in the netlist.

```
ERROR 1616: In command .IP_Protect, missing %s parameter or parameter
value.
```

The command contains one or several parameters that have no value, or a mandatory parameter has been forgotten.

```
ERROR 1618: Command .IP_Protect is not supported on that platform.
```

You have tried to run a circuit protected by ip_protect on a Windows platform. (Windows is not supported.)

```
ERROR 1619: Errors occur during command .IP_Protect: aborting the
simulation.
```

This appears at the end of the command if an error occurred before.

```
ERROR 1620: Fatal error while %s license %s from provider %s.\n+ Please
contact your provider for more details.
```

Eldo is unable to get a proper return value when trying to check-out or check-in the license.

```
ERROR 1621: Fatal error while loading shared library %s%s.\n+ Please
contact %s office.”
```

Eldo is unable to load the dynamic library.

```
Warning 1447: Unable to release %s license, feature %s\n+ %s
```

The license check-in failed.

```
Warning 1476: License message from Provider: %s – Feature: %s\n+
&quot;%s&quot;
```

The license tool has a specific message to deliver to the end user.

Appendix G
STMicroelectronics Models

Introduction

This section describes how to use the ST models inside Eldo, which in this case is called Eldo-ST.

How to Invoke Eldo-ST

Eldo-ST is invoked as follows:

```
eldo -stver ... cir_file_name
```

or by adding *before* the **.MODEL** cards the following:

```
.OPTION STVER
```

In case you want to disable the Eldo-ST mode, you can use the Eldo command-line argument:

```
eldo -nostver ... cir_file_name
```

or add *before* the **.MODEL** cards the following:

```
.OPTION NOSTVER
```

In all cases, the **.OPTION** takes precedence.

What Does it Change?

When Eldo-ST is invoked, model levels are automatically changed to use Eldo-ST in exactly the same way as ST-SPICE:

Table G-1. Eldo/Eldo-ST Model Levels

Component	Eldo	Eldo-ST
MOS	**.MODEL ... LEVEL**=18 or **LEVEL**=STMOS1	**.MODEL ... LEVEL**=1
	.MODEL ... LEVEL=19 or **LEVEL**=STMOS3	**.MODEL ... LEVEL**=3
Bipolar	**.MODEL ... LEVEL**=2	**.MODEL ... LEVEL**=1
Diode	**.MODEL ... LEVEL**=4	**.MODEL ... LEVEL**=1
	.MODEL ... LEVEL=5	**.MODEL ... LEVEL**=2
	.MODEL ... LEVEL=6	**.MODEL ... LEVEL**=3

Table G-1. Eldo/Eldo-ST Model Levels

Component	Eldo	Eldo-ST
JFET	`.MODEL ... LEVEL`=4	`.MODEL ... LEVEL`=1
	`.MODEL ... LEVEL`=5	`.MODEL ... LEVEL`=2
Resistor	`.MODEL ... LEVEL`=2	`.MODEL ... LEVEL`=1

All the standard Eldo levels may be used in Eldo-ST by adding the following in the `.MODEL` card:

```
MODTYPE=ELDO
```

STMicroelectronics Version of Eldo

Specially tuned device models have been developed for use when simulating STMicroelectronics designs. These STMicroelectronics models are available within the standard version of Eldo, for example as can be seen from the list of diode models available:

Table G-2. STMicroelectronics Model Selection

`LEVEL` Value	Model Name
1	Berkeley Level 1
2	Modified Berkeley Level 1
3	Fowler-Nordheim
4	**STMicroelectronics Level 1**
5	**STMicroelectronics Level 2**
6	**STMicroelectronics Level 3**

A special version of Eldo is also available which has been developed for use within the STMicroelectronics design kit. This version is identical to the standard version in every respect except that the STMicroelectronics device models occupy different levels within the lists of available device models. The different device model levels used within the STMicroelectronics version of Eldo are listed below:

Junction Diode Models

Table G-3. STMicroelectronics Junction Diode Model Selection

`LEVEL` Value	Model Name
1	**STMicroelectronics Level 1**
2	**STMicroelectronics Level 2**

Table G-3. STMicroelectronics Junction Diode Model Selection

LEVEL Value	Model Name
3	**STMicroelectronics Level 3**

MOSFET Models

Table G-4. STMicroelectronics MOSFET Model Selection

LEVEL Value	Model Name
1	**STMicroelectronics Level 1**
3	**STMicroelectronics Level 3**

Bipolar Junction Transistor Model

Table G-5. STMicroelectronics BJT Model Selection

LEVEL Value	Model Name
1	**STMicroelectronics Level 1**

Index

— Symbols —

— B —

— C —

— D —

— E —

— N —

— P —

— Q —

— R —

— S —

— T —

— U —

— V —

— W —

— X —

— Y —

— Z —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/terms_conditions/enduser

IMPORTANT INFORMATION

USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE. USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) between the company acquiring the license ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

1.1. To the extent Customer (or if and as agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will invoice separately. Unless provided with a certificate of exemption, Mentor Graphics will invoice Customer for all applicable taxes. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. Notwithstanding anything to the contrary, if Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under such orders in the event of default by the third party.

1.3. All products are delivered FCA factory (Incoterms 2000) except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all products delivered under this Agreement, to secure payment of the purchase price of such products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form; (b) for Customer's internal business purposes; (c) for the term; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of receiving support or consulting services, evaluating Software or

otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Software available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties excluding Mentor Graphics competitors provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") mean Mentor Graphics' proprietary syntaxes for expressing process rules. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or allow its use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code.

5.2. Customer may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it, whether by operation of law or otherwise ("attempted transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable transfer charges. Any attempted transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.3. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services for Software, Mentor Graphics will provide Customer with available updates and technical support for the Software which are made generally available by Mentor Graphics as part of such services in accordance with Mentor Graphics' then current End-User Software Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **LIMITED WARRANTY.**

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will meet Customer's requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under the applicable Order and does not renew or reset, by way of example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LICENSED AT NO COST; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **LIFE ENDANGERING APPLICATIONS.** NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH CUSTOMER'S USE OF SOFTWARE AS DESCRIBED IN SECTION 9. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INFRINGEMENT.**

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Software product infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, (a) replace or modify Software so that it becomes noninfringing, or (b) procure for Customer the right to continue using Software, or (c) require the return of Software and refund to Customer any license fee paid, less a reasonable allowance for use.

11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

11.4. THIS SECTION IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

12. **TERM.**

12.1. This Agreement remains effective until expiration or termination. This Agreement will immediately terminate upon notice if you exceed the scope of license granted or otherwise fail to comply with the provisions of Sections 2, 3, or 5. For any other material breach under this Agreement, Mentor Graphics may terminate this Agreement upon 30 days written notice if you are in material breach and fail to cure such breach within the 30 day notice period. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

12.2. Mentor Graphics may terminate this Agreement immediately upon notice in the event Customer is insolvent or subject to a petition for (a) the appointment of an administrator, receiver or similar appointee; or (b) winding up, dissolution or bankruptcy.

12.3. Upon termination of this Agreement or any Software license under this Agreement, Customer shall ensure that all use of the affected Software ceases, and shall return it to Mentor Graphics or certify its deletion and destruction, including all copies, to Mentor Graphics' reasonable satisfaction.

12.4. Termination of this Agreement or any Software license granted hereunder will not affect Customer's obligation to pay for products shipped or licenses granted prior to the termination, which amounts shall immediately be payable at the date of termination.

13. **EXPORT.** Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. Customer agrees that it will not export Software or a direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this section shall survive the termination of this Agreement.

17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of the Mentor Graphics intellectual property rights licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: This Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia (except for Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the Chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. All notices required or authorized under this Agreement must be in writing and shall be sent to the person who signs this Agreement, at the address specified below. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 090402, Part No. 239301