cadence®

Virtuoso® UltraSim Simulator User Guide

Product Version 7.2
May 2010

Contents

Preface

The *Virtuoso® UltraSim Simulator User Guide* is intended for integrated circuit designers who want to use the Virtuoso UltraSim™ Fast SPICE simulator to analyze the function, timing, power, noise, and reliability of their circuit designs.

This manual describes the following topics:

Licensing in Virtuoso UltraSim Simulator

Virtuoso UltraSim L and XL Product Levels

This section describes the Virtuoso UltraSim simulator L and XL product levels:

- **L** The L product level provides basic Virtuoso UltraSim simulation features
- **XL** The XL product level provides standard Virtuoso UltraSim simulation features, such as post layout simulation options, power network solver (UPS), netlist-based EMIR flow, `dx` simulation mode, and advanced circuit checks

Additional topics described in this section include feature details for each product level (see Simulator Product Level Features on page 23) and tracking token licenses (see Tracking Token Licenses on page 25).

The following documents give more information about Cadence token licenses and the FLEXlm™ license manager.

- *Cadence Installation Guide*
- *Cadence License Guide*
- *Virtuoso Software Licensing and Configuration Guide*

Using Different Simulator Product Levels

The Virtuoso UltraSim simulator product is available at two different levels: The most basic level is L and the next highest level is XL. The XL level provides additional simulation features capable of simulating more complex circuit designs.

Number	Product Level	Feature String
33400	Virtuoso® UltraSim Simulator L	`ULTRASIM_L`
33500	Virtuoso® UltraSim Simulator XL	`ULTRASIM`
90001/ 90002	Virtuoso® Multi Mode Simulation	`Virtuoso_Multi_mode_Simulation`

The Virtuoso UltraSim L product level requires the 33400 license or four tokens from the 90001/90002 license, whereas Virtuoso UltraSim XL requires the 33500 license or six tokens from the 90001/90002 license.

The license check out priority for the Virtuoso UltraSim simulator is in decreasing order of priority: `ULTRASIM` first, `ULTRASIM_L` next, and `Virtuoso_Multi_mode_Simulation` last.

You can change the order of priority using the `+lorder` command line option. For example,

```
+lorder ULTRASIM_L
+lorder ULTRASIM
+lorder ULTRASIM_L:ULTRASIM
+lorder Virtuoso_Multi_mode_Simulation
```

Notes

- You can only use `+lorder` to change the Virtuoso UltraSim simulator license check out procedure.
- For the L and XL product levels, you can check out licenses one per user, host, and display.
- To stop a license check out session, press `CTRL Z` on your keyboard.

Simulator Product Level Features

The *Virtuoso UltraSim Simulator User Guide* describes all of the Virtuoso UltraSim L and XL product level features. The following table provides a high level overview of the L and XL features.

Note: For more information about the L and XL product levels, refer to the Appendix A, "Virtuoso UltraSim L/XL Product Level Comparison Table."

Feature	UltraSim L	UltraSim XL
Netlist and model support (see "Netlist File Formats" on page 51 for details)	**X**	**X**
All circuit elements, transient sources, and simulator control statements are supported (see "Simulation Options" on page 155)	**X**	**X**
Simulation modes and accuracy settings (see "Simulation Modes and Accuracy Settings" on page 157)	**X**	**X**

Feature	UltraSim L	UltraSim XL
Waveform file format and resolution options (see “Waveform File Format and Resolution Options” on page 197)	X	X
All major device models are supported (see “Virtuoso UltraSim Simulator Features” on page 31)	X	X
Timing analysis and checks (see “Timing Analysis” on page 443)	X	X
Power analysis for all elements at the subcircuit and chip levels (see “Power Analysis” on page 419)	X	X
Interactive simulator debugging capability (see “Interactive Simulation Debugging” on page 335)	X	X
Basic simulator checks (see “Virtuoso UltraSim Advanced Analysis” on page 381)	X	X
Noise analysis (see “Active Node Checking” on page 382)	X	X
Virtuoso UltraSim C-Macromodel Interface (UCI) – see *Virtuoso UltraSim C-Macromodel Interface Reference*	X	X
Virtuoso UltraSim Waveform Interface (UWI) – see *Virtuoso UltraSim Waveform Interface Reference*	X	X
Digital vector file format (see “Digital Vector File Format” on page 613)	X	X
UltraSim-Verilog (see *Virtuoso Analog Design Environment L User Guide*)	X	X
Post layout simulation options and statistical reports		X
UltraSim power network solver (UPS)		X
Netlist-based EM/IR flow		X
Reliability age/agemos and control statements		X
Fast and local envelope simulation		X
Simulation mode `dx`		X
Bisection method		X
Advanced circuit checks		X
Virtuoso Unified Reliability Interface (URI)		X
Voltage regulator (VR) simulation		X

Tracking Token Licenses

You can use the lmstat UNIX shell command or the UltraSim log file to track token license activity.

UltraSim Log File

The UltraSim log file contains the following license check out information – successful checkout of:

- An ULTRASIM license
- An ULTRASIM_L license
- Four Virtuoso_Multi_mode_Simulation tokens
- Two Virtuoso_Multi_mode_Simulation tokens

The last log file output indicates an upgrade to the Virtuoso_Multi_mode_Simulation license (the Virtuoso UltraSim simulator first checks out four tokens and then two extra tokens incrementally, as needed).

lmstat Utility

The lmstat utility can be used to track token license activity. This utility reads the license.file and displays specific information when using the following options.

Option	Description
-a	Display all of the information
-c license_file	Use "license_file" as license file
-f [feature_name]	List usage information about specified (or all) features
-i [feature_name]	List information about specified (or all) features from the increment line in the license file
-S [DAEMON]	Display all users of DAEMONs licenses
-s [server_name]	Display status of all license files on server node(s)
-t timeout_value	Set connection timeout to "timeout_value"
-v	Display FLEXlm version, revision, and patch

`-old`	Allow communications with an old server that uses communications version 1.2 or earlier
`-help`	Prints specified message

Example

To use `lmstat`:

1. Type the following statement in a shell window.

   ```
   lmstat -c license.file -f "Virtuoso_Multi_mode_Simulation"
   ```

2. Review the output.

   In this example, the user (`wqin`) has started the `Virtuoso_Multi_mode_Simulation` license and is using six tokens.

   ```
   lmstat - Copyright (c) 1989-2006 Macrovision Europe Ltd. and/or Macrovision
   Corporation. All Rights Reserved.
   Flexible License Manager status on Thu 5/24/2007 16:40
   Users of Virtuoso_Multi_mode_Simulation:  (Total of 6 licenses issued;  Total
   of 6 licenses in use)
     "Virtuoso_Multi_mode_Simulation" v6.2, vendor: cdslmd
     floating license
        wqin usimlx100 unix:0 (v6.100) (usimlx100/5280 104), start Thu 5/24 16:39,
   4 licenses
        wqin usimlx100 unix:0 (v6.100) (usimlx100/5280 203), start Thu 5/24 16:39,
   2 licenses
   ```

For more information on `lmstat`, refer to the *Cadence License Manager* manual.

Related Documents for Virtuoso UltraSim Simulator

For additional information about the Virtuoso UltraSim simulator and related products, refer to the following manuals:

- *Virtuoso Analog Design Environment L User Guide* describes how to use the Virtuoso analog design environment (ADE) to simulate analog designs. The manual also includes important information about the Virtuoso UltraSim/ADE interface (Chapter 13) and UltraSimVerilog (Chapter 14).
- *Virtuoso RelXpert Reliability Simulator User Guide* describes the Virtuoso RelXpert simulator and how to characterize and extract reliability parameters, generate model files for the simulator, prepare the SPICE input netlist file for the simulator, and run and interpret simulation results.

- *Virtuoso UltraSim C-Macromodel Interface Reference* tells how the Virtuoso UltraSim C-macromodel Interface (UCI) supports the use of functional elements, described in C language, in the Virtuoso UltraSim simulator.
- *Virtuoso Unified Reliability Interface Reference* shows how the Virtuoso Unified reliability interface allows you to add your own reliability models to the Virtuoso UltraSim simulator and how the interface supports user-defined degradation models.
- *Virtuoso UltraSim Waveform Interface Reference* describes how to write Virtuoso UltraSim probe data and read probe data into the Virtuoso UltraSim simulator.
- *Virtuoso UltraSim Simulator What's New* introduces the new features for the Virtuoso UltraSim simulator release.
- *Virtuoso UltraSim Simulator Known Problems and Solutions* describes important Cadence change request records (CCRs) for the Virtuoso UltraSim simulator and tells you how to solve or work around these problems.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

Commands

`command_name [argument(s)]`

`argument types: keyword |` *`value`* `| tag = keyword | tag =` *`value`*

Table 1-1 Virtuoso UltraSim Argument Types

Argument Type	Definition
keyword	Keywords are the identifiers in a card that are defined by the Virtuoso UltraSim simulator.
value	Values are user-defined. These include elements names, node names, expected values, and value arguments to tags. They are shown in *italics* to emphasize that they are user-defined, as opposed to keywords and tags that are defined by the Virtuoso UltraSim simulator.
tag	Tags are identifiers in a card to which a value or keyword can be assigned. An example are the tags for element instance parameters (for example, MOSFET W, L, AS, AD). Tags can have values or keywords as arguments, as specified by the command syntax.

Table 1-2 Virtuoso UltraSim Symbol Types

Symbol Type	Definition
bar |	Represents the word `OR`, so you can choose between arguments.
ellipsis ...	Allows you to specify multiple arguments.
brackets [ ]	Indicates the enclosed argument is optional.
parentheses ( )	Indicates there is a choice between the enclosed arguments (two or more), and is only used when a command uses several groups of arguments.

In the following example, the statement specifies a `Cxx` capacitor, and `n1` and `n2` nodes. The `c` and `m` keywords have values assigned to them to specify the capacitance and multiplier factors, respectively (keywords are optional and are defined by square brackets). The values are displayed in *italics* to emphasize that the values are user-defined.

```
Cxx n1 n2 [c=value] [m=value]
```

In this Virtuoso Spectre format example, the `usim_opt speed` command expects a `1` or `2` as a value argument.

```
usim_opt speed=1|2
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.usim_opt speed`).

Syntax

- Numeric values in the control statement can be specified in decimal notation (xx.xx) or in engineering notation (x.xxe+xx).
- Values for *time* are specified in units of seconds. The key scale factor can be used by attaching a suffix `y` (year), `h` (hour), or `m` (minute).

  Note: Do not leave a space between the number and suffix (for example, 10m, 1e-5sec).
- Values for *current* are expected in units of A. The key scale factor can be used by attaching the suffix m=1e-3, u=1e-6, or n=1e-9.
- Values for *voltage* are expected in units of V. The key scale factor can be used by attaching the suffix m=1e-3, u=1e-6, or n=1e-9.
- Values for *length/width* are expected in units of meters.
- Values for *temperature* are expected in units of C (Celsius).
- The Virtuoso UltraSim simulator uses its default values if some of the control statements are not specified.

1

Introduction to Virtuoso UltraSim Simulator

The Virtuoso® UltraSim™ simulator is a fast and multi-purpose single engine, hierarchical simulator, designed to verify analog, mixed signal, memory, and digital circuits. The simulator can be used for functional verification of billion-transistor memory circuits and for high-precision simulation of complex analog circuits. Based on hierarchical simulation technology, the Virtuoso UltraSim simulator is faster and uses less memory than traditional circuit simulators, while maintaining near SPICE accuracy.

The Virtuoso UltraSim simulator supports all major netlist file formats and industry standard device models. It includes a comprehensive post-layout simulation solution and provides powerful deep-submicron analysis capabilities, including timing, power, noise, reliability, and IR drop analysis.

Virtuoso UltraSim Simulator Features

The main features of the Virtuoso UltraSim simulator include:

- Advanced transient pre- and post-layout simulation technology for analog, mixed signal, memory, and digital circuits delivering near SPICE accuracy, with significant performance acceleration over conventional SPICE, and virtually limitless capacity for hierarchically structured designs.
- 32- and 64-bit software available on Linux, Solaris, and IBM platforms (for detailed platform information, refer to http://support.cadence.com/wps/mypoc/cos?uri=deeplinkmin:DocumentViewer;src=wp;q=ProductInformation/LifeCycle/platform.html).
- Support of Virtuoso Spectre® and HSPICE netlist file formats, Verilog-A language, post-layout detailed standard parasitic format (DSPF) and standard parasitic exchange format (SPEF) netlist files, and structural Verilog® netlist files.
- Support of digital vector file format, and Verilog® value change dump (VCD) and extended VCD (EVCD) digital stimuli formats.

- SignalScan Turbo 2 (SST2), fast signal database (FSDB), parameter storage format (PSF), and waveform data format (WDF) generation.
- Superior RC reduction algorithms for post-layout simulation.
- Support of all major Virtuoso Spectre and HSPICE device models, including BSIM3, BSIM4, BSIMSOI, TFT, HVMOS, BJT, Mextram, Hicum, VBIC, and the flash memory cell model.
- Timing checks for setup and hold, rise and fall times, and pulse width.
- Power analysis at the element, subcircuit, and chip level.
- Design and device checks, including device voltage check, high impedance node analysis, DC leakage current analysis, and excessive device current check.
- Noise analysis, which monitors voltage overshoot (VO) and voltage undershoot (VU) effects on nodes.
- IR drop simulation using the Virtuoso UltraSim power network solver (UPS).
- Fast envelope analysis for high performance transient analysis of RF circuits.
- Reliability simulation, including hot carrier degradation (HCI), negative bias temperature instability (NBTI), aged simulation, and compatibility with Virtuoso RelXpert reliability simulator commands.
- Virtuoso UltraSim C-macromodel interface (UCI) for implementing user-specific analog or digital macromodels, such as PLL, memory block, analog to digital converter (ADC), and digital to analog converter (DAC).
- Virtuoso Unified reliability interface (URI) for implementing user-specific reliability models.
- Virtuoso UltraSim waveform interface (UWI) for customizing output of waveform formats.
- Integration into the Cadence analog design environment (ADE).
- Matlab toolbox to import PSF or SST2 data into MATLAB® (refer to the *Virtuoso Spectre Circuit Simulator RF Analysis User Guide* for more information).
- Standalone measurement tool to apply `.meas` to existing SST2 or FSDB waveform files.

Along with being the Cadence Fast SPICE transistor level simulator, the Virtuoso UltraSim simulator engine is used with the following Cadence tools:

- AMSUltra for Verilog/VHDL co-simulation with NCSIM.
- UltraSimVerilog for mixed signal co-simulation with VerilogXL.

- Virtuoso analog VoltageStorm option (VAVO) for power grid analysis of analog and mixed signal circuits.
- Virtuoso analog ElectronStorm option (VAEO) for electromigration analysis of analog and mixed signal circuits.
- VoltageStorm for power grid analysis of digital circuits and full chip designs.

Related Documents for Extended Analyses

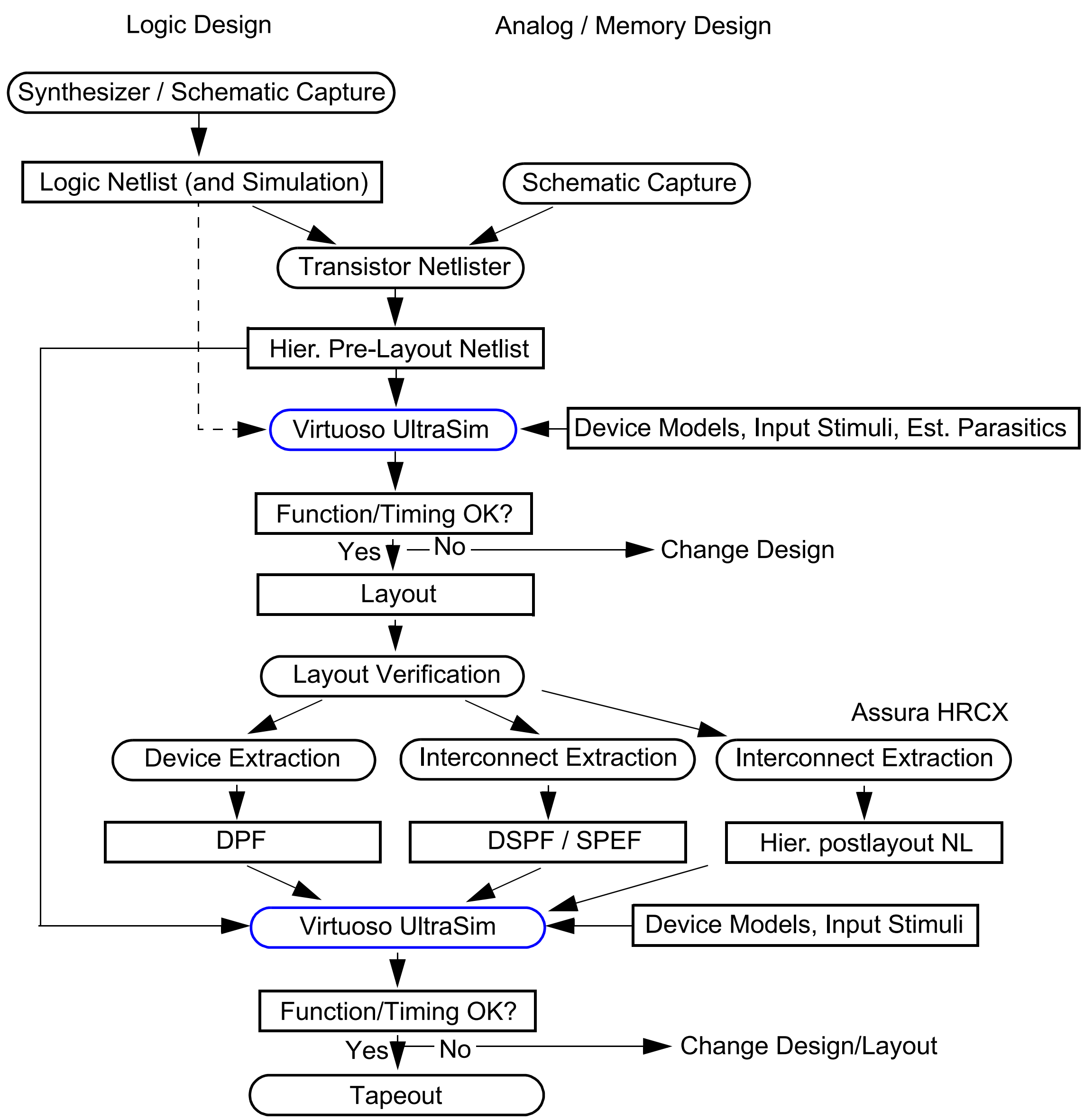
Refer to the following Cadence documentation for more information about these extended analyses:

- *Virtuoso AMS Designer Simulator User Guide* describes AMS UltraSim for Verilog/ VHDL co-simulation with NCSIM.
- *Virtuoso Analog Design Environment L User Guide* describes UltraSimVerilog and mixed signal co-simulation with VerilogXL.
- *Virtuoso Analog VoltageStorm and ElectronStorm User Guide* describes VAVO, VAEO, and other VoltageStorm analyses.

Virtuoso UltraSim Simulator in IC Design Flow

The Virtuoso UltraSim simulator can be used for pre- and post-layout simulation of analog, mixed signal, memory circuits, and logic designs. Figure 1-1 on page 34 shows how the simulator fits into the IC design flow.

Figure 1-1 Virtuoso UltraSim Simulator in IC Design Flow



Virtuoso UltraSim pre-layout simulation is used to verify design functionality and timing behavior, analyze the impact of submicron effects on the design, and to optimize the design before starting on the layout. Pre-layout simulation is based on a Spectre or SPICE netlist file, generated by schematic capture or synthesis tools, and also on device model files and input stimuli. An alternative is to input a synthesized or structural Verilog netlist file and the SPICE representation for all logic gates directly into the Virtuoso UltraSim simulator.

Virtuoso UltraSim post-layout simulation is used to verify circuit behavior after the layout design is completed. The simulation considers the effect of slightly changed device sizes, wire delays, and capacitive coupling created during the layout design, and allows the layout designer to optimize the layout design in regard to performance, power consumption, design margins, and robustness and reliability. The Virtuoso UltraSim simulator supports all major post-layout simulation flows, including DSPF/SPEF stitching, DPF backannotation, and the Cadence hierarchical extraction and simulation flow with Assura hierarchical resistor and capacitor extraction (HRCX).

Command Line Format

Running the Virtuoso UltraSim Simulator

The Virtuoso UltraSim simulator can be run from the command line by typing the following statement into a terminal window

```
ultrasim [-f]<circuit> [Options]
```

Note: You need to set the path to `your_install_dir`*/tools/bin* prior to running the Virtuoso UltraSim simulator.

Virtuoso UltraSim Simulator Options

Table 1-1 on page 35 lists the Virtuoso UltraSim simulator command line options.

Table 1-1 Command Line Options

Argument	Description
[`-f`] *`circuit`*	Circuit netlist filename (the netlist file can be compressed using `gzip` – see “Compressed Netlist File” on page 57 for more information)
`-h`	Prints the designated help message
`-info`	Prints system information
`-libpath path`	Loads the shared library
`-log`	Output messages are not copied to a file
`+log file`	Copies all messages to a file
`=log file`	Sends all messages to a file

Table 1-1 Command Line Options, *continued*

Argument	Description
`+lqtimeout value`	Specifies a duration (in seconds) for which the software should wait to check-out a license. When you set this option to `0`, the Virtuoso UltraSim simulator waits for a license until it is available. *Default*: 900 seconds **Note:** `+lqt` can be used as an abbreviation of `+lqtimeout`.
`+lreport`	Reports the number of required tokens in the log file. **Note:** `+lrpt` can be used as an abbreviation of `+lreport`.
`-raw rawDir`	Specifies the directory in which all parameter storage format (PSF) files are created
`-outdir outDir`	Specifies the directory in which all of the output files are created
`-outname` *`filename`*	Specifies the base filename which is used when files are created
`-format fmt`	Displays waveform data in `fmt` format (possible values for `fmt` include `psf`, `psfxl`, `sst2`, `fsdb`, or `wdf`; only one entry is allowed)
`-uwifmt name`	Specifies multiple waveform formats or user-defined output format [use a colon (`:`) as a delimiter to specify multiple formats]
`+rtsf`	Enables RTSF mode for all PSF files created and delivers improved viewing performance in the Virtuoso Visualization & Analysis (ViVA) tool
`+lsuspend`	Turns on license suspend/resume capability
`+lorder`	Checks licenses in a specific order (use `:` between license feature names when defining the order)
`-top subckt`	Creates a top level instance of the subcircuit
`-V`	Displays the Virtuoso UltraSim simulator version This option is case sensitive.
`-W`	Displays the Virtuoso UltraSim simulator subversion This option is case sensitive.
`-I dir`	Search `dir` directory for `.include` files
`-cmd cmdfile`	Command file for interactive simulation debugging

Table 1-1 Command Line Options, *continued*

Argument	Description
`-cmiconfig`	Read file for information used to modify existing compiled-model interface (CMI) configuration
`-i`	Invokes interactive shell
`-spectre`	Circuit netlist file in Virtuoso Spectre format
`-vlog Verilog_file`	Circuit netlist file in Verilog format
-mica	Circuit netlist file in Freescale® MICA format
`-csfe`	Disables simulation front end (SFE) parser shared with Spectre® simulator, and enables UltraSim front end (UFE) parser
`-r file`	Enables the static power grid solver
`-rout`	Enables the static power grid solver post-layout feature

Notes

- If the log file option is not specified, the Virtuoso UltraSim simulator automatically generates an output file named `circuit.ulog`.
- If `[+=]log` is specified, then the simulator always uses the option during simulation. This option only affects the name of the log file. If a path is not given for `[+=]log`, the final path for the log file follows the setting specified by the `-outdir` option. If `[+=]log` is not specified, the default `ulog` file follows the `-outdir` and `-outname` options.
- If `-raw` and `-outdir` are specified, `-raw` is overwritten by the simulator. All output files are placed into the directory specified in `-outdir`, unless `+log`, `usim_save`, or `model_lib` is used to specify the path for the corresponding files. A new directory is created if one does not already exist.
- If `-outname` is specified, all the output files using the netlist file name as a prefix are changed to the name defined in `-outname`.

Examples

In the following example, the Virtuoso UltraSim simulator writes the information into a log file named `circuit.log`.

```
ultrasim circuit.sp =log circuit.log
```

In the next example, the information is displayed on a standard output display device (same result if `-log` is not specified in the command).

```
ultrasim circuit.sp -log
```

In the next example, the Virtuoso UltraSim simulator writes the information into a log file named `circuit.log` and also displays it on a standard output display device.

```
ultrasim circuit.sp +log circuit.log
```

Waveform Post-Processing Measurement

```
ultrasim -readraw waveform <options> circuit
```

Description

The Virtuoso UltraSim simulator supports post-processing measurements based on waveform data from a prior simulation run. To perform measurements on an existing waveform file, add a `.measure` statement to the original netlist file and rerun the simulation using the `-readraw` statement (the regular simulation process is skipped). The post-processing measurement results are reported in `.pp.mt0` and `.pp.meas0` files. The default post-processing log file name is `.pp.ulog`.

Arguments

`circuit`	The filename of the circuit netlist file that contains the `.measure` statements. **Note:** The circuit needs to be the same circuit that generated the waveform being measured.
`-readraw waveform`	Specifies the name and location of the waveform file. The supported waveform formats are SST2 or FSDB. **Note:** The location of the waveform file can include the relative or absolute path.
`options`	The options used for a Virtuoso UltraSim simulation.

The `-readraw` statement can be used to perform measurements based on signals or expression probes.

Important

All basic signals used in the post-processing measurement need to be probed in the existing waveform file.

Examples

In the following example, the Virtuoso UltraSim simulator saves the `v(x1.out)` and `i(x1.out)` signals in the SST2 waveform `top.trn` file.

```
ultrasim -spectre top.sp
```

To perform a power calculation, the following measurement statement is added to the `top.sp` file.

```
.measure tran power avg v(x1.out)*i(x1.out) from=0ns to=1us
```

Note: You can also put `.measure` statements in a file (for example, `measure.txt`) and include it in the `top.sp` file.

To start the post-processing measurement, use the following statement.

```
ultrasim -readraw top.trn -spectre top.sp
```

The measurement results are reported in the `top.pp.mt0` and `top.pp.meas0` files.

Virtuoso UltraSim 64-Bit Software

To run Virtuoso UltraSim 64-bit software,

1. Use the `-debug3264 -V` command to check your system configuration:

   ```
   $your_install_dir/tools/bin/ultrasim -debug3264 -V
   ```

   You can use the information to verify if the 64-bit version is applicable to your platform, if the 64-bit software is installed, and whether or not it is selected.

2. Install the Virtuoso UltraSim 64-bit software to the same location as your 32-bit software.

3. Verify that all required software patches are installed by running `checkSysConf` (system configuration checking tool script). The script is located in your local installation of Cadence software:

   ```
   $your_install_dir/tools/bin/checkSysConf MMSIM7.0
   ```

   The script is also available on the Cadence Online Support system.

4. Set the `CDS_AUTO_64BIT` environment variable `{all|none|"list"|include:"list"|exclude:"list"}` to select 64-bit executables.

- **`all`** invokes all applications as 64-bit.

  The list of available executables is located at:

  `$`*`your_install_dir`*`/tools/bin/64bit`

- **`none`** invokes all applications as 32-bit.

- **`"list"`** invokes only the executables included in the list as 64-bit.

  `"list"` is a list of case-sensitive executable names delimited by a comma (`,`), semicolon (`;`), or colon (`:`).

- **`include:"list"`** invokes all applications in the list as 64-bit.

- **`exclude:"list"`** invokes all applications as 64-bit, except the applications contained in the list.

Note: If `CDS_AUTO_64BIT` is not set, the 32-bit executable is invoked by default.

Example

```
setenv CDS_AUTO_64BIT ultrasim
setenv CDS_AUTO_64BIT "exclude:si"
```

5. Launch the executables through the wrapper.

   All 64-bit executables are controlled by a wrapper executable. The wrapper invokes the 32-bit or 64-bit executables depending on how the `CDS_AUTO_64BIT` environment variable is set, or whether the 64-bit executable is installed. The wrapper also adjusts the paths before invoking the 32-bit or 64-bit executables. The wrapper you use to launch the executables is located at *`your_install_dir`/tools/bin.*

   Note: Do not launch the executables directly from the *`your_install_dir`/tools/bin/64bit* or *`your_install_dir`/tools/bin/32bit* directory.

   ### Example

   `$your_install_dir``/tools/bin/ultrasim`

Virtuoso UltraSim Simulator Configuration File

The Virtuoso UltraSim simulator supports a common configuration file called `ultrasim.cfg`, enabling you to set the default options for the simulator. This file can be located at three levels. The Virtuoso UltraSim simulator searches for the `ultrasim.cfg` file in the following locations, in this order:

1. Working directory of the netlist file.
2. Home directory (`$HOME`).

3. Virtuoso UltraSim simulator installation directory (`$ULTRASIM_ROOT`).

This allows the Virtuoso UltraSim simulator to be configured by you, by the site, or by the project. The Virtuoso UltraSim simulator processes only the first `ultrasim.cfg` it reads. That is, the `ultrasim.cfg` in the netlist file directory overwrites the `ultrasim.cfg` in `$HOME` and the Virtuoso UltraSim simulator installation directories. The `ultrasim.cfg` file can contain the following types of commands:

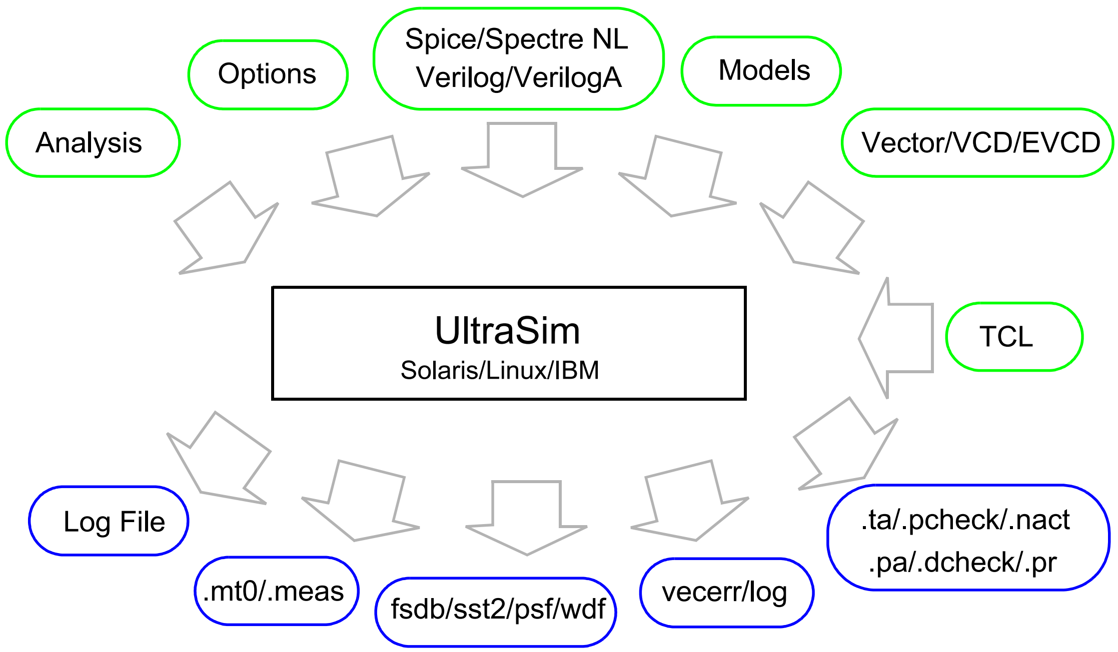
- All Virtuoso UltraSim options (Virtuoso Spectre or HSPICE syntax).
- Virtuoso Spectre `tran`, `options`, and `save` commands.
- HSPICE `.tran`, `.options`, and `.probe` commands.

Note: If Virtuoso Spectre syntax is used in the `ultrasim.cfg` file, `simulator lang=spectre` needs to be specified at the beginning of the file.

Virtuoso UltraSim Simulator Input/Output Files

The Virtuoso UltraSim simulator recognizes Virtuoso Spectre, SPICE, Verilog-A, and structural Verilog netlist file formats. Figure 1-2 on page 42 gives an overview of the input and output data required for simulation with the Virtuoso UltraSim simulator. The simulator also supports all major Spectre and SPICE device models (see Chapter 2, “Netlist File Formats,” for more details). Digital vector file format and VCD/EVCD stimuli are described in Chapter 13, “Digital Vector File Format” and Chapter 14, “Verilog Value Change Dump Stimuli.” The Virtuoso UltraSim simulation options for optimizing simulation accuracy and performance are located in Chapter 3, “Simulation Options.”

Figure 1-2 Virtuoso UltraSim Simulator Input/Output Files Diagram



In addition to a log file, the Virtuoso UltraSim simulator creates several output files that contain waveforms, measurements, and analysis results. Each output file has an extension followed by a number. The output files are defined in Table 1-2 on page 42 below.

Table 1-2 Output Files

Extension	Format	Content
`actnode`	ASCII	Active node check from `acheck` (Virtuoso Spectre format)
`chk_capacitor`	ASCII	Prints capacitor statistics into a log file
`chk_resistor`	ASCII	Prints resistor statistics into a log file
`dcheck`	ASCII	Device voltage report from `dcheck` (Virtuoso Spectre format)
`elemcut`	ASCII	Contains elements that were cut because their value was less than the specified threshold
`fsdb`	binary	Fast signal database (FSDB) waveform file (`wf_format=fsdb`; waveform viewer: nWave)
`icmd`	ASCII	Interactive mode command history
`ilog`	ASCII	Interactive mode log file

Table 1-2 Output Files, *continued*

Extension	Format	Content
`meas`	ASCII	Results from `.meas` (SPICE format)
`mt`	ASCII	Results from `.meas` (SPICE format)
`nact`	ASCII	Node activity report from `usim_nact` (Virtuoso Spectre format)
`nodecut`	ASCII	Cut nodes
`pa`	ASCII	Element and subcircuit power report from `usim_pa` (Virtuoso Spectre format)
`para_rpt`	ASCII	Prints subcircuit parameters into a report file
`part_rpt`	ASCII	Prints partition and node connectivity analysis results into a report file
`pcheck`	ASCII	Report for excessive current, DC path leakage current, and high impedance node checks (Virtuoso Spectre format)
`pr`	ASCII	Partitioning and node connectivity from `usim_report` (Virtuoso Spectre format)
`print`	ASCII	Table printout from `.print`
`rpt_chkmosv`	ASCII	MOSFET bias voltage check log file
`rpt_chknmosb`	ASCII	NMOSFET drain/source junction check log file
`rpt_chknmosvgs`	ASCII	MOSFETs with n-type channels check log file
`rpt_chkpar`	ASCII	Parameter check log file
`rpt_chkpmosb`	ASCII	PMOSFET drain/source junction check log file
`rpt_chkpmosvgs`	ASCII	MOSFETs with p-type channels check log file
`rpt_chksubs`	ASCII	Substrate check log file
`rpt_maxleak`	ASCII	Maximum DC leakage paths report
`ta`	ASCII	Setup, hold, pulse width, and timing edge violations from `usim_ta` (Virtuoso Spectre format)
`tran`	binary	Parameter storage format (PSF) waveform file (`wf_format=psf`; waveform viewers: Virtuoso Visualization and Analysis, and AWD)

Table 1-2 Output Files, *continued*

Extension	Format	Content
trn/dsn	binary	SignalScan Turbo (SST2) waveform file (wf_format=sst2; default format; waveform viewers: SimVision and Virtuoso Visualization and Analysis)
ulog	ASCII	Log file (default if -/=/+log command line option not specified)
vecerr	ASCII	Vector and VCD/EVCD check errors
veclog	ASCII	Vector and VCD/EVCD check results
wdf	binary	WDF waveform file (wf_format=wdf; waveform viewer: Sandworks)

For mt files, the number following the file extension corresponds to the .alter number. For example, if there are two .alter blocks in the netlist file, the mt files are called .mt0, .mt1, and .mt2. The naming convention is HSPICE compatible.

For all other output files, the number corresponds to the number of times the transient analysis was run. For example, if the main netlist file block specifies two different temperatures in the .temp command card, and there is an .alter block that modifies the original .temp command card and specifies new temperature values, then the transient analysis for this netlist file needs to be run three times. All output files generated from the first run would not have a number after the extension. For example, the FSDB file is named circuit.fsdb. The output files generated from the second and third runs are named circuit.fsdb1 and circuit.fsdb2, respectively.

Waveform Name Syntax

The Virtuoso UltraSim simulator generates the waveform output file with the hierarchical signal name (except for PSF format). The signal names have default syntax for SPICE and Virtuoso Spectre netlist file formats.

SPICE Netlist File Syntax

If the input netlist file contains SPICE format, then the generated waveform names use the following syntax:

```
<output-type>(<node>)
```

The `<output-type>` syntax can be either V, I, X0, or any other output type supported by the Virtuoso UltraSim simulator, and `<node>` is the name of the node specified in the probe statement.

For current probes, the output type (for example, i1 or i2) is based on the branch of the element or node specified in the probe statement.

Virtuoso Spectre Netlist File Syntax

If the input netlist file contains Virtuoso Spectre format, then the generated waveform names use the following syntax:

- `<node>` for voltage probes
- `<elem>:<num>` for current probes
- `<node>:<output-type>` for all other output types, where `<node>` is the name of the node specified in the probe statement and `<num>` is the branch of the element or node for which the waveform is needed (default `<num>` is `1`)

SPICE and Virtuoso Spectre Netlist File Syntax

If the input netlist file contains SPICE and Virtuoso Spectre syntax, then the default for the waveform name is Virtuoso Spectre syntax. You can override the default behavior of the waveform syntax by using the `wf_spectre_syntax` option.

For example,

```
.usim_opt wf_spectre_syntax=1
```

Setting this option to `1` in the input netlist file forces the output waveform names to follow Virtuoso Spectre syntax, independent of whether the input netlist file is in SPICE, Virtuoso Spectre, or both formats. If the option is set to `0`, the output waveform names follow SPICE syntax, independent of the input netlist file format.

Note: The Virtuoso UltraSim simulator waveform output generated in the analog design environment (ADE) is always in Spectre syntax, irrespective of the input netlist file format or the `wf_spectre_syntax` option.

Virtuoso UltraSim Return Codes

The Virtuoso UltraSim simulator supports two types of return codes: `0` and `1`. A return of `0` indicates the simulation was successfully completed and a return of `1` indicates the simulation failed.

Error and Warning Messages

The Virtuoso UltraSim simulator issues error, warning, and information messages when problems are encountered during circuit design simulation.

- An *error* message reports a condition that the simulator cannot resolve (if the error is severe, it may cause the simulator to stop completely).
- A *warning* message reports an unusual condition that does not adversely affect the simulation.
- An *info* message presents information that does not fall into either of the other message categories (info messages are generally used to give the status about a process that is running).

Creating Tutorial Directories

The Virtuoso UltraSim simulator tutorials provide examples to help you get started using the simulator. Running the tutorials is recommended to obtain hands-on experience using the Virtuoso UltraSim simulator features and options. There are four categories of tutorials in the `examples` directory of the Virtuoso UltraSim simulator installation:

- UltraSim_Workshop – Virtuoso UltraSim simulator standalone and Virtuoso UltraSim simulator in the Cadence analog design environment (ADE).
- usim_ade – Virtuoso UltraSim simulator in ADE.
- Usim_Verilog – Virtuoso UltraSim simulator and Verilog-XL co-simulation.

  Note: All of the Virtuoso UltraSim simulator ADE and Verilog-XL examples can be run in the IC 5.0.33 USR3 and 5.0.41 or later releases (fast envelope analysis in ADE requires 5.0.33 USR4 or 5.1.41 USR1).

- USIM_NetlistBased_EMIR_Flow – Virtuoso UltraSim simulator netlist-based electromigration (EM) and IR drop analysis flow.

  Note: This flow is based on OpenAccess (OA), and IC 5.1.41 USR3 or IC 6.1 and higher is required.

UltraSim_Workshop

The `UltraSim_Workshop` tutorial includes 16 examples, covering the most important features of the Virtuoso UltraSim simulator (that is, Virtuoso UltraSim standalone and Virtuoso UltraSim/ADE examples).

To run `UltraSim_Workshop`:

1. Create a directory called `ultrasim_workshop`.
2. Copy the `pll.tar.gz`, `mult.tar.gz`, and `sp_mult_ade.tar.gz` files to the `ultrasim_workshop` directory from *`ultrasim_install_dir`*`/tools/ultrasim/examples/UltraSim_Workshop/` as shown below:

```
cd ultrasim_workshop
cp -r ultrasim_install_dir/tools/ultrasim/examples/UltraSim_Workshop/* .
```

3. Untar the `pll.tar.gz` and `mult.tar.gz` files.

```
gzip -cd pll.tar.gz | tar xvf -
gzip -cd mult.tar.gz | tar xvf -
gzip -cd sp_mult_ade.tar.gz | tar xvf -
```

4. Follow the instructions in the *UltraSim_workshop.pdf* document (located in the `./doc/` directory).

usim_ade

The `usim_ade` tutorial contains a complete, step-by-step example which describes how to run key Virtuoso UltraSim simulator options in ADE.

If you are using version:

- IC5033USR2, IC5033USR3, or IC5141, copy the files from *`ultrasim_install_dir`*`/tools/ultrasim/examples/usim_ade/5033USR2_USR3_5141/*`.
- IC5033USR4 or IC5141USR1, copy the files from *`ultrasim_install_dir`*`/tools/ultrasim/examples/usim_ade/5033USR4_5141USR1/*`.

To run `usim_ade`

1. Create a directory called `ultrasim_ade`.
2. Copy the `usimADE_tut.tar.gz` file to the `ultrasim_ade` directory from *`ultrasim_install_dir`*`/tools/ultrasim/examples/usim_ade/` as shown below:

```
cd ultrasim_ade
cp -r ultrasim_install_dir/tools/ultrasim/examples/usim_ade/
5033USR2_USR3_5141/* .
```

Note: The updated ADE tutorials are located in the `5141USR2` directory.

3. Untar the `usimADE_tut.tar.gz` file.

```
gzip -cd usimADE_tut.tar.gz | tar xvf -
```

4. Follow the instructions in the *UltraADE_tut.pdf* document.

Usim_Verilog

The `Usim_Verilog` tutorial contains a mixed signal example for Virtuoso UltraSim simulator and Verilog-XL co-simulation.

If you are using version:

- IC5033USR2, IC5033USR3, or IC5141, copy the files from *`ultrasim_install_dir`*`/tools/ultrasim/examples/Usim_Verilog/5033USR2_USR3_5141/*`.
- IC5033USR4 or IC5141USR1, copy the files from *`ultrasim_install_dir`*`/tools/ultrasim/examples/Usim_Verilog/5033USR4_5141USR1/*`.

To run `Usim_Verilog`

1. Create a directory called `verimix_usim`.
2. Copy the `UsimVerilog_tut.tar.gz` file to the `verimix_usim` directory from *`ultrasim_install_dir`*`/tools/ultrasim/examples/Usim_Verilog/` as shown below:

```
cd verimix_usim
cp -r ultrasim_install_dir/tools/ultrasim/examples/Usim_Verilog/
5033USR2_USR3_5141/* .
```

Note: The updated UltraSimVerilog tutorials are located in the `5141USR2` directory.

3. Untar the `UsimVerilog_tut.tar.gz` file.

```
gzip -cd UsimVerilog_tut.tar.gz | tar xvf -
```

4. Follow the instructions in the *UsimVerilog_tut.pdf* document.

USIM_NetlistBased_EMIR_Flow

The `USIM_NetlistBased_EMIR_Flow` tutorial contains a netlist-based EM/IR flow example for the Virtuoso UltraSim simulator.

To run `USIM_NetlistBased_EMIR_Flow`

1. Create a directory called `USIM_EMIR`.

2. Copy the `USIM_EMIR_FLOW_OA.tar.gz` file to the `USIM_EMIR` directory from *`ultrasim_install_dir`*`/tools/ultrasim/examples/USIM_NetlistBased_EMIR_Flow` as shown below.

```
cd USIM_EMIR
```

```
cp -r ultrasim_install_dir/tools/ultrasim/examples
USIM_NetlistBased_EMIR_Flow/* .
```

3. Untar the `USIM_EMIR_FLOW_OA.tar.gz` file.

```
gunzip USIM_EMIR_FLOW_OA.tar.gz | tar xvf -
```

4. Follow the instructions in the `USIM_EMIR_FLOW_workshop.pdf` document.

2

Netlist File Formats

The Virtuoso® UltraSim™ simulator recognizes SPICE, Virtuoso Spectre®, Verilog®-A, and structural Verilog netlist files. This chapter describes the syntax rules, elements, and command cards supported by the Virtuoso UltraSim simulator, and also describes the known limitations.

Supported Netlist File Formats

You can use the Virtuoso UltraSim simulator to simulate Virtuoso Spectre and HSPICE (registered trademark of Synopsys, Inc.) netlist files. Virtuoso Spectre and HSPICE netlist file formats follow different case sensitivity and naming convention rules. These differences affect interpretation of the netlist file devices, elements, parameters, topology, and simulator option statements. When setting up a Virtuoso UltraSim simulation, it is important to understand the differences in syntax rules, so the simulation produces the correct results.

The following netlist file formats are supported:

- Virtuoso Spectre on page 51
- HSPICE on page 52
- Mixed Virtuoso Spectre and HSPICE on page 54
- Structural Verilog on page 56

Note: Cadence recommends using either the Spectre or HSPICE languages exclusively in a netlist file.

Virtuoso Spectre

If the netlist file is in Virtuoso Spectre format, you need to set the `-spectre` command line option or add the `.scs` extension to your top-level netlist file. In this case, the Virtuoso UltraSim simulator behaves the same as Virtuoso Spectre, and applies Virtuoso Spectre naming conventions and case sensitivity to:

- Node and hierarchy names
- Keywords
- Parameters
- Units of measurement

The Virtuoso UltraSim simulator also uses Virtuoso Spectre default values for model and simulation setup.

All of the Virtuoso UltraSim simulator options are available in Virtuoso Spectre syntax format, so you can define the options in a Virtuoso Spectre netlist file. The most common Virtuoso UltraSim simulator option is `usim_opt`. The Virtuoso Spectre syntax is located under the *Spectre Syntax* heading in each Virtuoso UltraSim option section.

Example

Spectre Syntax:

```
simulator lang=spectre
usim_opt sim_mode=ms speed=6 postl=2
usim_opt sim_mode=a inst=i1.i2.vco1
usim_pa chk1 subckt inst=[i1.i2] time_window=[1u 5u]
dcheck chk1 vmos model=[tt] inst=[i1.*] vgsu=1.0 vgsl=0.5 probe=1
usim_report resistor type=distr rmin=0 rmax=20
```

HSPICE

If you simulate a design that uses HSPICE format in the netlist file, you need to use Virtuoso UltraSim format without the `-spectre` command line option or `.scs` extension. In this case, the Virtuoso UltraSim simulator behaves the same as HSPICE, and applies HSPICE naming conventions and case insensitivity to:

- Node and hierarchy names
- Keywords
- Parameters
- Units of measurement

The Virtuoso UltraSim simulator also uses HSPICE default values for model and simulation setup.

All of the Virtuoso UltraSim simulator options are available in SPICE syntax format, so you can define the options in a HSPICE netlist file. The most common Virtuoso UltraSim simulator option is `.usim_opt`. SPICE syntax is located under the *SPICE Syntax* heading of each Virtuoso UltraSim option section.

Example

HSPICE Syntax:

```
.usim_opt sim_mode=ms speed=6 postl=2
.usim_opt sim_mode=a inst=x1.x2.vco1
.usim_pa chk1 subckt inst=[x1.x2] time_window=[1u 5u]
.dcheck chk1 vmos model=[tt] inst=[x1.*] vgsu=1.0 vgsl=0.5 probe=1
.usim_report resistor type=distr rmin=0 rmax=20
```

Table 2-1 on page 53 compares Virtuoso UltraSim simulator option syntax rules for HSPICE and Virtuoso Spectre netlist files.

Table 2-1 HSPICE and Virtuoso Spectre Syntax Comparison Table

HSPICE Syntax	Virtuoso Spectre Syntax
HSPICE Language Rules: ■ Case insensitive ■ HSPICE compatible models ■ Global parameter passing (`parhier=global`) ■ `temp=tnom` and `tnom=25` ■ Built-in parameters: `time`, `temp`, `hertz` ■ `0`, `gnd`, `gndl`, and `ground` are all global ground nodes, and are reported as `v(0)`	**Virtuoso Spectre Language Rules:** ■ Case sensitive ■ Virtuoso Spectre compatible models ■ Local parameter passing ■ `temp=27` and `tnom=27` ■ Built-in parameters: `time`, `temp`, `tnom`, `scale`, `scalem`, `freq` ■ First node in global statement

Command Line:
`ultrasim file`
(except `file.scs`)
 	
All files are required to be in HSPICE format.
	Command Line:
	`ultrasim -spectre file`
	or
	`ultrasim file.scs`
	
	All files with a `.scs` extension are assumed to be in Virtuoso Spectre format and all other files are assumed to be in HSPICE format.
	
	If files contain Spectre syntax, but do not use the `.scs` extension, they need to contain `simulator lang=spectre` at the beginning of the file to inform the Virtuoso UltraSim simulator that the content is in Virtuoso Spectre format.

Table 2-1 HSPICE and Virtuoso Spectre Syntax Comparison Table, *continued*

HSPICE Syntax	Virtuoso Spectre Syntax
HSPICE Netlist and Models: `.model, .subckt, .ends, .end, ...`	**Virtuoso Spectre Netlist and Models:** `model`, `subckt`, `end`, `simulator lang=...`, `inline`, ...
HSPICE Analysis and Options: `.tran`, `.probe`, `.op`, `.meas`, `.ic`, `.data`, `.options`, `.temp`, `.alter`, ...	**Virtuoso Spectre Analysis and Options:** `tran`, `options`, `ic`, `save`, `altergroup`
Verilog-A: `.hdl`	**Verilog-A:** `ahdl_include`
Virtuoso UltraSim Structural Verilog: `.vlog_include`	**Virtuoso UltraSim Structural Verilog:** `vlog_include`
Virtuoso UltraSim Vector Stimuli: `.vec`, `.vcd`, `.evcd`	**Virtuoso UltraSim Vector Stimuli:** `vec_include`, `vcd_include`, `evcd_include`
Virtuoso UltraSim Options and Analyses: `.usim_opt`, `.usim_ta`, `.usim_nact`, `.pcheck`, `.dcheck`, `.acheck`, `.usim_save`, `.usim_restart`, `.usim_report`, ...	**Virtuoso UltraSim Options and Analyses:** `usim_opt`, `usim_ta`, `usim_nact`, `pcheck`, `dcheck`, `acheck`, `usim_save`, `usim_restart`, `usim_report`, ...
Important The Virtuoso UltraSim simulator behaves like HSPICE (cannot be mixed with Virtuoso Spectre syntax).	*Important* The Virtuoso UltraSim simulator behaves like Virtuoso Spectre, and Virtuoso Spectre and SPICE syntax can be mixed using `lang simulator=spice|spectre lookup=spice|spectre`. Cadence recommends using a Virtuoso Spectre-only format netlist file.

Mixed Virtuoso Spectre and HSPICE

In some cases, a mix of the Virtuoso Spectre and HSPICE languages is required. For example, when a design uses Virtuoso Spectre format in the netlist file and the device models are only available in HSPICE format. Since HSPICE does not support using mixed languages in a netlist file, the Virtuoso UltraSim/Spectre `-spectre` command can be used to simulate the mixed format file.

Note: Use the `simulator lang=spectre|spice` command to switch between the Virtuoso Spectre and HSPICE languages in the netlist file.

Example

Spectre/HSPICE Mixed Syntax:

```
simulator lang=spectre
global 0 2
v1 (2 0) vsource dc=2.0
...
mos (4 2 0 0) nmos1
tran1 tran tstop=100n
...
simulator lang=spice lookup=spectre
.model nmos1 nmos level=49 version = 3.1 ...
...
simulator lang=spectre
```

In this example, the `lang=spectre|spice` command defines the language rules for the section of the netlist file that follows the statement until the next `lang=spectre|spice` command is issued or the end of the file is reached.

The `lookup=spectre` portion of the `simulator=spice` command specifies that all node, device, and instance names follow the Virtuoso Spectre naming convention. This is necessary for proper mapping between nodes, devices, and instances in the Virtuoso Spectre and HSPICE sections of the netlist file.

Wildcard Rules

The Virtuoso UltraSim simulator allows you to use wildcards (`*`) in the `.probe`, `.lprobe`, `.ic`, `.nodeset`, and `save` statements, as well as in all of the Virtuoso UltraSim scopes, options, and checking features.

The following rules apply to wildcards:

- A single asterisk (`*`) matches any string, including an empty string and a hierarchical delimiter
- A question mark (`?`) matches any single character, including a hierarchical delimiter

Examples

- `.probe v(*)` matches all signals on all levels (for example, `vdd`, `x1.net5`, `x1.x2.sa`, and `x1.x2.x3.net7`).
- `.probe v(*) depth=2` matches all signals in the top two levels (for example, `vdd`, `x1.net5`, but not `x1.x2.sa`).
- `.probe v(*t)` matches all top level signals ending with `t` (for example, `vnet`, `m_t`, `senst`, but not `x1.net`).

- `.probe v(*.*t)` matches all signals on all levels ending with `t` (for example, `vnet`, `m_t, x1.net`, and `x1.x2.x3.at`).
- `.probe v(net?8)` matches all signals on all levels (for example, `net08`, `net88`, and `net.io8`).
- `save * depth=2` saves all node voltages on the top level and one level below (for example, `net12`, `i1.net28`, and `x1.net9`, but not `x1.x2net8`).

Structural Verilog

Spectre Syntax

```
vlog_include "file.v" supply0=gnd supply1=vdd insensitive=no|yes
```

SPICE Syntax

```
.vlog_include "file.v" supply0=gnd supply1=vdd insensitive=no|yes
```

Description

The Virtuoso UltraSim simulator supports structural Verilog netlist files for verification of digital circuits. This approach is typically used for standard cell designs, where the Verilog netlist file is generated by synthesis tools, and SPICE subcircuits are available for all standard cells.

The most common approach is to use a top level SPICE file which contains the analysis statement, probes, measures, and simulation control statements, and also calls one or multiple Verilog netlist files. The Verilog netlist files contain calls of the basic cells, which are available in the SPICE netlist file.

To activate the Verilog parser, use the Virtuoso UltraSim simulator `-vlog` option in the command line (for more information, refer to "Command Line Format" on page 35). You can also include Verilog files by using the `vlog_include` statement(s).

The Virtuoso UltraSim simulator reads the structural Verilog file `file.v`. The keywords are `supply0` and `supply1`. The `supply0` keyword must be set to the ground node used in the Verilog subcircuit and `supply1` must be set to the power supply node. If `insensitive=yes`, the Verilog netlist file is parsed case insensitive. If `insensitive=no`, it is parsed case sensitive. The default value is `no`.

Note: If the name of a module called by SPICE contains uppercase letters (for example, `top_MODULE`), then set `insensitive=yes` to use the `vlog_include` statement.

Unsupported Structural Verilog Features

The Virtuoso UltraSim simulator does not support the following structural Verilog features:

- Multi-bit expression (for example, `3’b1`)
- Arrayed instances
- defparam
- `trireg`, `triand`, `trior`, `tri0`, `tri1`, `wand`, and `wor` nets
- Strength and delay
- Generated instances
- User-defined procedures (UDPs)
- Attributes

The Virtuoso UltraSim simulator resolves bus signals into individual signals when reading Verilog netlist files. The bus notation can be set using the buschar option and either `<>` or `[]`.

The simulator also supports bus node mapping in structural Verilog. When instantiating the Verilog module in an analog netlist file, port mapping can only be based on the order of the signal name definitions. The bus node in the Verilog netlist file is expanded in the analog netlist file. When invoking an analog cell in a Verilog netlist file, port mapping can be based on the order of the signal definitions or names. For name mapping, the bus notation in the analog netlist file can be set using the vlog_buschar option.

Compressed Netlist File

The Virtuoso UltraSim simulator can read a compressed top-level netlist or included files (`.include`, `.lib`, `.vec`, `.vcd`, `spf`, and `spef`). The compressed netlist or included file needs to be compressed using `gzip` (`.gz` file extension).

Note: A compressed included file can be nested within another compressed file.

Example

```
ultrasim circuit.sp.gz
```

is a compressed `circuit.sp` file called `circuit.sp.gz`. A compressed `model.gz` file is nested within the `circuit.sp` file:

```
.include model.gz
```

Supported Virtuoso Spectre Model Features

Virtuoso Spectre

The Virtuoso UltraSim simulator recognizes circuit topologies in Virtuoso Spectre circuit simulator format for operating point and transient analysis. It does not support other analysis types, such as DC, AC, and noise.

For a detailed list and description of the related Virtuoso Spectre constructs, refer to the *Virtuoso Spectre Circuit Simulator User Guide*.

The Virtuoso UltraSim simulator shares all device model interfaces with Virtuoso Spectre and therefore supports the same models:

- **MOSFET:** bsim3v3, bsim4, sp32, b3soipd, bsimsoi (versions 2.23 and higher, including version 4.0), bsim1, bsim2, bsim3, bta silicon-on-insulator (btasoi), enz-krummenacher-vittoz (ekv), mos0, mos1, mos2, mos3, mos6, mos7, mos8, high-voltage mos (hvmos), poly thin film transistor (psitft), alpha thin film transistor (atft), and ldmos.

  Note: For more information about the sp32 model, refer to the “Surface Potential Based Compact MOSFET Model (spmos)” chapter in the *Virtuoso Simulator Components and Device Models Reference*.

- **BJT:** bipolar junction transistor (bjt), bht, hetero-junction bipolar transistor (hbt), vertical bipolar inter-company (vbic), mextram (bjt503 and bjt504)
- **Diode:** diode
- **JFET:** junction field effect transistor (jfet)

The Virtuoso UltraSim simulator also supports the same set of customer models supported by Virtuoso Spectre, such as the Philips, ST, Infineon, and Nortel models.

Unsupported Virtuoso Spectre Features

The following Virtuoso Spectre components, analysis and controls, and features are not supported by the Virtuoso UltraSim simulator:

Components	
assert	ucccs
core	uccvs
node	uvccs
paramtest	uvcvs
quantity	winding

Analysis and Controls		
ac	pdisto	qpxf
alter	pnoise	set
check	psp	shell
dc	pss	sp
dcmatch	pxf	stb
envlp	qpac	sweep
montecarlo	qpnoise	tdr
noise	qpsp	xf
pac	qpss	

Features	
checkpoint	sens
export	spectremdl
param_limits	spectrerf

Virtuoso UltraSim device models are implemented using the Cadence compiled-model interface (CMI). You can also implement proprietary device models with CMI. For more information about installing and compiling device models using CMI, refer to Appendix C, "Using Compiled-Model Interface" in the *Virtuoso Spectre Circuit Simulator User Guide*.

Verilog-A

Spectre Syntax

```
ahdl_include “file.va”
```

SPICE Syntax

```
.hdl “file.va”
```

Description

Verilog-A behavioral models can be applied to Virtuoso Spectre netlist files using the `ahdl_include` statement, and in HSPICE netlist files using `.hdl`.

Verilog-A behavioral language is used to model the behavior of analog design blocks. The Virtuoso UltraSim simulator supports Verilog-A behavioral language formats and provides a parser which is compatible with the Virtuoso Spectre simulator parser. Refer to the *Cadence Verilog-A Language Reference* for more information about supported language constructs.

Unsupported Verilog-A Features

The Virtuoso UltraSim simulator does not support the following Verilog-A features:

- Potential/flow attributes
- Disciplines (except electrical)
- Power consumption calculations

Note: The Fast SPICE technology used in the Virtuoso UltraSim simulator, such as representative device models, partitioning, and hierarchical simulation, cannot be applied to Verilog-A behavioral models (Verilog-A dominated designs will not show a performance advantage with the Virtuoso UltraSim simulator when compared to conventional SPICE tools).

Supported HSPICE Model Features

Syntax Rules

The Virtuoso UltraSim simulator syntax rules are similar to HSPICE syntax rules.

- The maximum line length is 1024 characters.
- The maximum length of a word, such as `name`, is 1024 characters.
- The following characters are not allowed in any name: `{}`, `()`, `"`, `'`, `=`, `;`, `:`
- Any expression must be in single quotation marks '*`expression`*'.
- SPICE and Virtuoso UltraSim simulator commands are prefixed by a period (`.`) character.
- Element instances begin with a particular character based on the element type. For example, metal oxide semiconductor field-effect transistor (MOSFET) names begin with `m`. See "Supported HSPICE Devices and Elements" on page 63 for more information.
- Commands and instances can be continued across multiple lines by using the `+` sign in the beginning of each continuation line. Names, parameters, and arguments cannot be continued across multiple lines.
- Virtuoso RelXpert reliability simulator commands start with the `*relxpert:` prefix. Virtuoso RelXpert command cards can be continued across multiple lines by using the `+` continuation character (that is, `*relxpert: +`).

  Note: You need to include a space after the colon (`:`) in the `*relxpert:` prefix.
- Comment lines must begin with an `*` or `$` sign. Comments can be written in a new line, or after the end of an instance or command on the same line.
- Comment lines and/or blank lines are allowed between continuation lines of a multi-line command or instance.
- Virtuoso UltraSim simulator is case insensitive and converts all names to lower case, except for filenames.
- Hierarchical node names are allowed for elements, but elements cannot be given hierarchical names.
- Virtuoso UltraSim simulator can recognize abbreviated names for SPICE commands, as long as the abbreviated name yields a unique command. For example, `.tr` or `.tra` can be recognized as `.tran`.

The `.t` command does not work because `.temp` card also begins with `.t`.

Unit Prefix Symbols

The following unit prefix symbols can be applied to any numerical quantities:

- a = A = 1.0e-18
- F = f = 1.0e-15
- G = g = 1.0e9
- K = k = 1.0e3
- M = m = 1.0e-3
- N = n = 1.0e-9
- P = p = 1.0e-12
- T = t = 1.0e12
- U = u = 1.0e-6
- X = x = MEG = meg = 1.0e6
- Y = y = 1.0e-24
- Z = z = 1.0e-21

Supported HSPICE Devices and Elements

The Virtuoso UltraSim simulator supports the following HSPICE devices and elements:

- Bipolar Junction Transistor on page 64
- Capacitor on page 67
- Current-Controlled Current Source (F-Element) on page 69
- Current-Controlled Voltage Source (H-Element) on page 71
- Diode on page 73
- Independent Sources on page 75
- JFET and MESFET on page 77
- Lossless Transmission Line (T-Element) on page 79
- Lossy Transmission Line (W-Element) on page 80
- MOSFET on page 83
- Mutual Inductor on page 86
- Resistor on page 87
- Self Inductor on page 89
- Voltage-Controlled Current Sources (G-Elements) on page 91
- Voltage-Controlled Voltage Source (E-Elements) on page 95

The following sections provide a brief description of the elements supported by the Virtuoso UltraSim simulator.

Bipolar Junction Transistor

```
Qxxx nc nb ne [ns] model_name [area = area_val] [areab = areab_val]
     + [areac = areac_val] [m = mval] [dtemp = dtemp_val]
```

Description

The Virtuoso UltraSim simulator ignores the initial condition parameters specified for bipolar junction transistors (BJTs). The BJTs supported by the simulator are listed below.

BJT level 1	Gummel-Poon model
BJT level 2	BJT Quasi-Saturation model
BJT level 6	Mextram model
BJT level 8	HiCUM model
BJT level 9	VBIC99 model

Arguments

`nc, nb, ne`	Collector, base, and emitter terminal node names.
`ns`	Substrate terminal node name; can also be set in a BJT model with the `bulk` or `nsub` parameters.
`model_name`	BJT model name.
`area` = `area_val`	Emitter area multiplying factor (default = 1.0).
`areab` = `areab_val`	Base area multiplying factor (default = `area`).
`areac` = `areac_val`	Collector area multiplying factor (default = `area`).

`m = mval`	Multiplier to indicate how many elements are in parallel.
	■ The `m` multiplier is used in a netlist file instance call (default value is `m=1`)
	■ The `m` parameter is set in the .subckt or .param statements
	Note: The multiplier is different from the `m` parameter.
	For example
	`.subckt R_only up down m=5`
	`R00 up down 100 m='m+2'`
	`.ends`
	where `m=5` in the `.subckt` statement is the `m` parameter definition and `'m+2'` is an expression that is dependent on the `m` parameter in the subcircuit. The `m` located to the left of the equal (`=`) sign is a multiplier that is evaluated at 7 (5+2) to indicate seven resistors with a value of 100 ohms are in parallel.
	Notes
	■ The `m` parameter must be preset before use. The `m` multiplier does not need to be preset because it has a default value of 1.
	■ If `m=5` is not defined in the example above, the simulation fails and produces an "undefined parameter" error message.

`dtemp = dtemp_val`	The difference between the element temperature and the circuit temperature in Celsius (default = 0.0).

Examples

In the following example

```
q001 c b a npn
```

defines a `npn BJT q001` with its collector, base, and emitter connected to nodes `c`, `b`, and `a`, respectively.

In the next example

```
q002 5 8 19 6 pnp area = 1.5
```

defines a `pnp BJT q002` with its collector, base, emitter, and substrate connected to nodes `5`, `8`, `19`, and `6`, respectively, and has an emitter factor of 1.5.

Capacitor

```
Cxx n1 n2 [model_name] [c = capacitance] [tc1 = val] [tc2 = val]
     + [scale = val] [m = mval] [w = val] [l = val] [dtemp = val]
```

or

```
Cxx n1 n2 poly c0 c1 ... [options shown above]
```

Description

If the instance parameter tags (such as `c`, `tc1`, and `tc2`) are not used, the arguments must be arranged in the same order as shown in the first syntax statement (see above). Otherwise, the instance arguments can appear in any order. In the second syntax statement, capacitance is determined by a polynomial function to be `c = c0 + c1 * v + c2 * v *v + ...,` where `v` is the voltage across the capacitor.

Arguments

`n1, n2`	Terminals of the capacitor.
`model_name`	Model name of the capacitor.
`c` = *`capacitance`*	Capacitance at room temperature. It can be a numerical value (in farads) or ■ An expression with parameters and functions of node voltages ■ Branch currents of other elements ■ Time, frequency, or temperature The argument is optional if a model name is specified.
`m = mval`	Element multiplier used to simulate multiple parallel capacitors (default = 1).
`tc1` = *`val`*	First-order temperature coefficient for the capacitor.
`tc2` = *`val`*	Second-order temperature coefficient for the capacitor.
`scale` = *`val`*	Scaling factor; scales capacitance by its value (default = 1.0)

`dtemp` = *`val`*	Temperature difference between the element and the circuit in Celsius (default = 0.0)
`l` = *`val`*	Capacitor length in meters (default = 0.0)
`w` = *`val`*	Capacitor width in meters (default = 0.0)
`c0, c1, ...`	Coefficients of the polynomial form for the capacitance (if none exists, zero is used)

Examples

In the following example

```
c001 1 0 5f
```

defines a capacitor connected to nodes `1` and `0`, with a capacitance of 5e-15 farad.

In the next example

```
c002 1 0 '1.5e-12*v(5)*time' tc1 = 0.001 tc2 = 0
```

defines a capacitor connected to nodes `1` and `0`, with a capacitance depending on the voltage of node `5` and time.

In the next example

```
c003 1 0 poly 1 0.5 scale = 1e-12
```

defines a capacitor connected to node `1` and `0` with a capacitance determined by

```
c = [1 + 0.5 v (1,0) ] * le-12
```

In the next example

```
c004 1 0 c=10p M=5
```

defines five capacitors in parallel, measuring 10 picofarads, and connected to nodes `1` and `0`.

Current-Controlled Current Source (F-Element)

Linear

```
Fxx n+ n- [cccs] vn1 gain [max = val] [min = val]
     + [scale = val] [tc1 = val] [tc2 = val] [abs = 1] [ic = val] [m=val]
```

Piece-Wise Linear

```
Fxx n+ n- [cccs] pwl(1) vn1 [delta = val] [scale = val]
     + [tc1 = val] [tc2 = val] [m = val] x1,y1 x2,y2 ... [ic = val] [m=val]
```

Polynomial

```
Fxx n+ n- [cccs] poly(ndim) vn1 [... vnndim]
     + [tc1 = val] [tc2 = val] [scale = val] [max = val]
     + [min = val] [m = val] [abs = 1] p0 [p1 ...] [ic = val] [m=val]
```

Delay Element

```
Fxx n+ n- [cccs] delay vn1 td = val [m=val]
     + [tc1 = val] [tc2 = val] [scale = val] [npdelay = val]
```

Description

Defines four forms of current-controlled current sources (CCCSs): Linear, piece-wise linear, polynomial, and delay element.

Arguments

`n+`, `n-`	Positive and negative terminals of the controlled source
`cccs`	Keyword for current-controlled current source (`cccs` is reserved for HSPICE and cannot be used as a node name)
`gain`	Current gain
`poly`	Keyword for polynomial dimension function (default is a one-dimensional polynomial)
	Note: `Ndim` must be a positive number.
`pwl`	Keyword for a piece-wise linear function
`max = val`	Maximum output current (default = `undefined`; no maximum set)
`min = val`	Minimum output current (default = `undefined`; no minimum set)

`m = mval`	Element multiplier to simulate multiple replication elements in parallel (default = `1`)
`tc1 = val`	First order temperature coefficient for `cccs`
`tc2 = val`	Second order temperature coefficient for `cccs`
`scale = val`	Scaling factor which scales capacitance by its value (default = 1.0)
`ic = val`	Estimate of IC initial condition for the controlling currents, measured in amps (default = 0.0).
`abs = 1`	Output is an absolute value, if `abs = 1`
`vn1`	Names of voltage sources through which the controlling current flows
`x1,...`	Controlling current through the *vn1* source (specify *x* values in increasing order)
`y1,...`	Corresponding output current values of *x*.
`delay`	Keyword for the `delay` element **Note:** `delay` is a reserved word that cannot be used as a node name.
`td = val`	Specifies the propagation delay for the macro model (subcircuit) process

Examples

In the following example

```
F001 1 0 VC 5 max=+3 min=-3
```

defines a `cccs` `F001`, connected to nodes 1 and 0 with a current value of I(F001)=5*I(VC). The current gain is 5, maximum current is limited to 3 amps, and minimum current is limited to -3 amps.

In the next example

```
F002 1 0 poly VC 1M 1.3M
```

defines a polynomial `cccs` `F002`, connected to nodes 1 and 0, and with a current value of I(F002)=1e-3+1.3e-3*I(VC).

In the next example

```
F003 1 0 delay VC td=7n scale=2
```

defines a delayed `cccs` `F003`, connected to nodes 1 and 0.

Current-Controlled Voltage Source (H-Element)

Linear

```
Hxx n+ n- [ccvs] vn1 transresistance [max = val] [min = val]
     + [scale = val] [tc1 = val] [tc2 = val] [abs = 1] [ic = val]
```

Piece-Wise Linear

```
Hxx n+ n- [ccvs] pwl(1) vn1 [delta = val] [scale = val]
     + [tc1 = val] [tc2 = val] x1, y1, x2, y2 ... [ic = val]
```

Polynomial

```
Hxx n+ n- [ccvs] poly(ndim) vn1 [... vnndim]
     + [tc1 = val] [tc2 = val] [scale = val] [max = val]
     + [min = val] [abs = 1] p0 [p1 ...] [ic = vals]
```

Delay Element

```
Hxx n+ n- [ccvs] delay vn1 td = val
     + [tc1 = val] [tc2 = val] [scale = val] [npdelay = val]
```

Description

Defines four forms of current-controlled voltage sources (CCVSs): Linear, piece-wise linear, polynomial, and delay element.

Arguments

n+, *n-*	Positive and negative terminals of the controlled source
`ccvs`	Keyword for current-controlled voltage source (`ccvs` is reserved for HSPICE and cannot be used as a node name)
transresistance	Current to voltage conversion factor
`poly`	Keyword for polynomial dimension function (default is a one-dimensional polynomial) **Note:** *Ndim* must be a positive number.
`pwl`	Keyword for a piece-wise linear function
`max =` *val*	Maximum output voltage (default = `undefined`; no maximum set)
`min =` *val*	Minimum output voltage (default = `undefined`; no minimum set)

`m = mval`	Element multiplier to simulate multiple replication elements in parallel (default = `1`)
`tc1 = val`	First order temperature coefficient for `ccvs`
`tc2 = val`	Second order temperature coefficient for `ccvs`
`scale = val`	Scaling factor which scales capacitance by its value (default = 1.0)
`ic = val`	Estimate of IC initial condition for the controlling currents, measured in amps (default = 0.0).
`abs = 1`	Output is an absolute value, if `abs = 1`
`vn1`	Names of voltage sources through which the controlling current flows
`x1,...`	Controlling current through the *vn1* source (specify *x* values in increasing order)
`y1,...`	Corresponding output current values of *x*.
`delay`	Keyword for the `delay` element **Note:** `delay` is a reserved word that cannot be used as a node name.
`td = val`	Specifies the propagation delay for the macro model (subcircuit) process

Example

In the following example

```
H001 1 0 VC 10 max=+10 min=-10
```

defines a `ccvs` `H001`, connected to nodes 1 and 0, with a voltage value of V(H001)=10*I(VC). The current to voltage gain is 5, maximum voltage is limited to 10 volts, and minimum voltage is limited to -10 volts.

Diode

Level = 1

```
Dxxx  n+ n- model_name [area = area_val] [pj = pj_val] [wp = wp_val]
     + [lp = lp_val] [wm = wm_val] [lm = lm_val] [m = mval]
     + [dtemp = dtemp_val]
```

Level = 2

```
Dxxx  n+ n- model_name [w = wval] [l = lval] [wp = wp_val] [lp = lp_val]
     + [ic = val] [m = mval]
```

Level = 3

```
Dxxx  n+ n- model_name [w = wval] [l = lval] [wp = wp_val] [lp = lp_val]
     + [wm = wm_val] [lm = lm_val] [m = mval] [dtemp = dtemp_val]
```

Description

Defines a diode with terminal connections, model, and geometries. The `[area,pj]` or `[w,l]` format can be used to specify the diode area. Other instance parameters have the same meaning for both formats. Initial conditions are ignored for diode elements. Diode model levels 1-3 are supported by the Virtuoso UltraSim simulator. The diode capacitance is modeled as an equivalent linear capacitor between the terminals.

The diodes supported by the simulator are listed below.

Level 1	Geometric, junction diode model
Level 2	Fowler-Nordheim model
Level 3	Non-geometric, junction diode model
Level 4	Philips juncap model

Arguments

`n+, n-`	Positive and negative terminals of a diode, respectively.
`model_name`	Model name for the diode.

`area` = *`area_val`*	Area of diode without units for `level=1` and m^2 for `level=3` (default = 1.0). Default value can be overridden from diode model. If unspecified, it is calculated from the width and length specifications: `area` = `l`*`w`.
`pj` = *`pj_val`*	Periphery of junction without units for `level=1` and in meters for `level=3` (default = 0.0). Default value can be overridden from diode model. If unspecified, it is calculated from the width and length specifications: `pj` = 2*(`l`+`w`).
`w = `*`wval`*	Width of junction in meters (default = 0.0).
`l` = *`lval`*	Length of junction in meters (default = 0.0).
`wp` = *`wp_val`*	Width of polysilicon capacitor in meters (default = 0.0).
`lp` = *`lp_val`*	Length of polysilicon capacitor in meters (default = 0.0).
`wm` = *`wm_val`*	Width of metal capacitor in meters (default = 0.0).
`lm` = *`lm_val`*	Length of metal capacitor in meters (default = 0.0).
`dtemp` = *`dtemp_val`*	The difference between the element temperature and the circuit temperature in Celsius (default = 0.0)
`m = mval`	Element multiplier (default = 1).

Examples

In the following example

```
d001 p n diode1
```

defines a diode named `d001` connected between nodes `p` and `n`. The diode model is `diode1`.

In the next example

```
d002 5 10 diode2 area = 1.5
```

defines a diode named `d002` connected between nodes `5` and `10`. The diode model is `diode2` and the PN junction area is 1.5.

Independent Sources

Voltage Source

```
Vxxx  n+ n- [dc_func] [tran_func]
```

Current Source

```
Ixxx  n+ n- [dc_func] [tran_func]
```

Description

Defines a voltage or current source. The direct current (DC) or one form of the transient functions is required, and only one form of the transient functions is allowed for each source. If a DC and a transient function coexist, the DC function is ignored even in a DC analysis. See "Supported HSPICE Sources" on page 98 for a description of these source functions.

Arguments

`n+, n-`	Positive and negative terminals respectively
`dc_func`	DC function specifying a voltage or a current
`tran_func`	A form of transient function (see "Supported HSPICE Sources" on page 98 for details)
`m` = *`m_val`*	Element multiplier (default = 1)

Examples

In the following example

```
v001 5 0 4.5
```

defines a voltage source between nodes `5` and `0` with a constant voltage of 4.5.

In the next example

```
v002 in gnd pulse( 0 4.5 100n 2n 2.5n 20n 25n )
```

defines a pulse voltage source between nodes `in` and `gnd`.

In the next example

```
i001 4 0 0.001
```

defines a current source between nodes `4` and `0` with a constant current 0.001A.

JFET and MESFET

```
Jxxx ndrain ngate nsource [nbulk] model_name [area=area] [w=width]
     + [l=length] [off] [ic=vdsval, vgsval] [m=val] [dtemp=val]
```

or

```
Jxxx ndrain ngate nsource [nbulk] model_name [area=area] [w=width]
     + [l=length] [off] [vds=vdsval] [vgs=vgsval] [m=val] [dtemp=val]
```

Description

Defines a JFET or metal semiconductor field effect transistor (MESFET) with terminal connections, models, and geometries. The required fields are the drain, gate, source nodes, and model name.

The JFET and MESFET models supported by the simulator are listed below.

Level 1	SPICE model
Level 2	Modified SPICE model
Level 3	SPICE compatible MESFET model

Arguments

`Jxxx`	JFET or MESFET element name.
	Note: Arguments must begin with `J`.
`ndrain`	Drain terminal node name.
`ngate`	Gate terminal node name.
`nsource`	Source terminal node name.
`nbulk`	Bulk terminal node name (optional).
`model_name`	Field effect transistor (FET) model name.
`area`	Area multiplying factor in units of square meters (default=1.0).
`w`	FET gate width in meters.
`l`	FET gate length in meters.

`off`	Sets initial condition to `off` for this element in DC analysis (default=`on`).
`ic=`*`vdsval`*, *`vgsval`*	Initial internal drain source voltage (`vds`) and gate source voltage (`vgs`).
`m`	Multiplier used to simulate multiple JFETs and MESFETs in parallel. Setting `m` affects all currents, capacitances, and resistances (default=1.0).
`dtemp`	The difference between the element temperature and the circuit temperature in Celsius (default=0.0)

Examples

In the following example

```
J001 ndrain ngate nsource jfet
```

defines a JFET with the name `J001` that has its drain, gate, and source connected to nodes `ndrain`, `ngate`, and `nsource`, respectively.

In the next example

```
J002 nd ng ns jfet area=100u
```

Defines a JFET with the name `J002` that has its drain, gate, and source connected to nodes `nd`, `ng`, and `ns`, and the area is 100 microns.

Lossless Transmission Line (T-Element)

`T`*`xx`* `in ref_in out ref_out z0 =` *`z0val`* `td =` *`tdval`* `[l =` *`length`*`]`

Description

Defines the lossless transmission line.

Arguments

`in`, `out`	Input and output node names
`ref_in`, `ref_out`	Ground reference node names for input and output signals
`z0` = *`z0val`*	Characteristic impedance in ohms
`td` = *`tdval`*	Transmission delay time in second/meter
`l` = *`length`*	Transmission line length in meters (default = 1)

Example

```
T1 1 r1 2 r2 z0 = 100 td = 1n l = 1
```

Defines a lossless transmission line connected to nodes `1` and `2`, and reference nodes `r1` and `r2`. The transmission line is one meter long, with an impedance of 100 ohms and a delay of 1 ns per meter.

Lossy Transmission Line (W-Element)

```
Wxx in1 in2 … inN ref_in out1 out2 … outN ref_out <rlgcmodel=name | rlgcfile=name>
     n=val l=val
```

Description

Defines the multi-conductor lossy frequency-dependent transmission line or W-element.

Arguments

`in1 … inN`	Node names for the near-end input signal terminals
`ref_in`	Node name for the near-end reference terminal
`out1 … outN`	Node names for the far-end signal terminals
`ref_out`	Ground reference node name for the far-end output terminal
`n`	Number of signal conductors, excluding the reference conductor
`l`	Length of the transmission line in meters
`rlgcmodel`	Name of the resistance, inductance, conductance, and capacitance (RLGC) model
`rlgcfile`	Name of the external file with RLGC parameters

Examples

In the following example

```
w1  1  r1  2  r2 rlgcmodel=t1_model n=1 l=0.5
.model  t1_model w modeltype=rlgc n=1
+ Lo = 3e-7 Co = 1e-10 Ro = 10 Go = 0 Rs = 1e-03  Gd = 1e-13
```

`w1` is a lossy transmission line connected to nodes 1 and 2, and reference nodes `r1` and `r2`. It has a length of 0.5 m and its electrical characteristic is specified by the RLGC model `t1_model`.

In the next example

```
w2  1  2  3  r1  4  5  6  r2 rlgcmodel=t3_model n=3 l=0.2
.model t3_model w modeltype=rlgc n=3
  +    Lo =
```

```
+    1e-6
+    2e-7 3e-6
+    4e-8 5e-7 6e-6
+    Co =
+    1e-11
+    -2e-12 3e-11
+    -4e-13 -5e-12 6e-11
+    Ro =
+    40
+    0 40
+    0 0 40
+    Go =
+    1e-4
+    -2e-5 3e-4
+    -4e-6  -5e-5 6e-4
+    Rs =
+    1e-3
+    0  1e-3
+    0 0 1e-3
+    Gd =
+    1e-13
+    -2e-14 3e-13
+    -4e-15 -5e-14 6e-13
```

`w2` is a three-conductor lossy transmission line with a length of 0.2 m. Its electrical characteristic is specified by the RLGC matrix model `t3_model`.

In the next example

```
w3   1   2   3   r1   4   5   6   r2 rlgcfile = tline.dat n=3 l=0.1
```

w3 is a lossy transmission line connected to nodes 1, 2 and 3, and reference nodes r1 and r2. It has a length of 0.1 m and its electrical characteristic is specified by the RLGC model `tline.dat` file.

Format of the model file `tline.dat`:

```
* The first number specifies the number of conductors.
3
* Lo =
1e-6
2e-7  3e-6
4e-8   5e-7   6e-6
* Co =
```

```
1e-11
-2e-12   3e-11
 -4e-13 -5e-12   6e-11
* Ro =
40
0  40
0  0   40
* Go =
1e-4
 -2e-5 3e-4
 -4e-6  -5e-5 6e-4
* Rs =
1e-3
0  1e-3
0  0   1e-3
* Gd =
1e-13
 -2e-14  3e-13
 -4e-15  -5e-14   6e-13
```

specifies the electrical characteristics of the `w3` lossy transmission line by the external model file `tline.dat`.

MOSFET

```
Mxx ndrain ngate nsource [nbody] model_name [l = length] [w = width]
     + [ad = ad_val] [as = as_val] [pd = pd_val] [ps = ps_val]
     + [nrd = nrd_val] [nrs = nrs_val] [rdc = rdc_val] [rsc = rsc_val]
     + [m = mval] [dtemp = dtemp_val] [geo = geo_val]
```

or

```
Mxx ndrain ngate nsource [nbody] model_name [length] [width] [ad] [as] [pd] [ps]
```

Description

Defines a metal oxide semiconductor (MOS) transistor with terminal connections, model, and geometries. Besides the element name, only the model name and the drain, gate, and source nodes are required. Initial conditions can be specified, but are ignored by the Virtuoso UltraSim simulator.

The second syntax is used in conjunction with `.options wl`, which changes the order so that width appears before length. The second syntax requires the instance parameters to be listed in the order given above. If more than six instance parameters are listed, an error is issued by the Virtuoso UltraSim simulator.

The MOSFET models supported by the simulator are listed below.

Level 49 and Level 53	BSIM3v3 versions 3.0, 3.1, 3.2, 3.21, 3.22, 3.23, 3.24, and 3.30
Level 54	BSIM4 versions 4.0, 4.1, 4.2, 4.3, 4.4, and 4.5
Level 69	PSP model
Level 57 and Level 59	BSIM3SOI versions 2.2, 2.21, 2.22, 2.23, 3.0, 3.1, and 3.2 (Level 59 is a SOI FD model and the Virtuoso UltraSim simulator only supports this model in `s` mode)
Level 70	BSIMSOI 4.0
Level 61	RPI a-Si TFT model
Level 62	RPI Poli-Si TFT model versions 1 and 2
HVMOS (Level 101)	Cadence proprietary high-voltage MOS model
Level 50	Philips MOS9 model
Level 63	Philips MOS11 model

EKV (Level 55)	-

Arguments

`ndrain, ngate, nsource`	Drain, gate, and source terminals of the MOS transistor, respectively.
`nbody`	Bulk terminal node name; set in a MOS model with parameter `bulk`.
`model_name`	Name of the model for the transistor.
`l` = *`length`*	Channel length of the transistor in meters.
`w` = *`width`*	Channel width of the transistor in meters.
`ad` = *`ad_val`*	Drain diffusion area.
`as` = *`as_val`*	Source diffusion area.
`pd` = *`pd_val`*	Drain diffusion perimeter.
`ps` = *`ps_val`*	Source diffusion perimeter.
`nrd` = *`nrd_val`*	Number of squares for drain diffusion.
`nrs` = *`nrs_val`*	Number of squares for source diffusion.
`rdc` = *`rdc_val`*	Additional drain resistance, units of ohms. This value overrides the rdc setting in the MOS model specification (default = 0.0).
`rsc = `*`rsc_val`*	Additional source resistance, units of ohms. This value overrides the RSC setting in the MOS model specification (default = 0.0).
`m` = *`mval`*	Multiplier to simulate multiple MOSFETs in parallel (default=1).
`dtemp` = *`dtemp_val`*	The difference between the element temperature and the circuit temperature in Celsius (default = 0.0).
`geo` = *`geo_val`*	Source/drain sharing selector for MOS model parameter value acm = 3 (default = 0.0).

Examples

In the following example

```
m001 1 2 3 nmos
```

defines a MOS transistor with name `m001` and model name `nmos`. The drain, gate, and source are connected to nodes `1`, `2`, and `3`, respectively. The bulk terminal is defined in the N-channel metal oxide semiconductor (NMOS) model, or the default value 0 is used. `l` and `w` are chosen as default values in this case.

In the next example

```
m002 a b c d nmos l = 0.2u w = 1u
```

defines a MOS transistor with the name `m002` and model name `nmos`. The drain, gate, source, and bulk are connected to nodes `a`, `b`, `c`, and `d`, respectively. `l` is 0.2 microns and `w` is 1 micron.

Mutual Inductor

`K`*`xx`* `L`*`yy`* `L`*`zz`* `[k =` *`coupling`*`]`

Description

Defines a mutual inductor, where `Lyy` and `Lzz` are inductors. Other HSPICE mutual inductor formats are not supported.

Arguments

`L`*`yy`*`, L`*`zz`*	Mutually coupled inductors.
`k` = *`coupling`*	Coefficient of mutual coupling. `k` is a unitless number with a magnitude greater than 0 and less than or equal to 1. If `k` is negative, the direction of coupling is reversed.

Example

```
K1 L1 L2 0.1
```

Defines a mutual inductor with a coefficient of 0.1 between inductor `L1` and inductor `L2`.

Resistor

```
Rxx n1 n2 [model_name] [[r =] val] [tc1 = val] [tc2 = val]
     + [scale = val] [m = val] [dtemp = val] [l = val] [w = val]
     + [c = val]
```

Description

Defines a linear resistor or wire element. If a resistor model is specified, the resistance value is optional. If the instance parameter tags (`r`, `tc1`, and `tc2`) are not used, the values must be ordered as shown above.

Arguments

`n1, n2`	Resistor terminal nodes.
`model_name`	Model name of the resistor.
`r` = *`val`*	Resistance value in ohms at room temperature. It can be a numerical value or ■ An expression with parameters and functions of node voltages ■ Branch currents of other elements ■ Time, frequency, or temperature
`m = mval`	Multiplier to simulate multiple resistors in parallel (default = 1).
`tc1` = *`val`*	First-order temperature coefficient for the resistor.
`tc2` = *`val`*	Second-order temperature coefficient for the resistor.
`scale` = *`val`*	Scaling factor; scales resistance and capacitance by its value (default = 1.0).
`dtemp` = *`val`*	Temperature difference between the element and the circuit in Celsius (default = 0.0).
`l` = *`val`*	Resistor length in meters (default = 0.0).
`w = `*`val`*	Resistor width in meters (default = 0.0).
`c` = *`val`*	Parasitic capacitance connected from node n2 to the bulk node (default = 0.0).

Examples

In the following example

```
r001 1 0 50
```

defines a resistor named `r001` with 50 ohms resistance connected between nodes `1` and `0`.

In the next example

```
r002 x y 150 tc1 = 0.001 tc2 = 0
```

defines a 150 ohm resistor `r002` between nodes `x` and `y` with temperature coefficient `tc1` and `tc2`.

In the next example

```
r003 1 2 ‘5*v(5, 6)^2+f(x,y)’
```

defines a resistor connected to nodes `1` and `2`, with a resistance depending on voltage deference between nodes `5` and `6` in the given expression.

Self Inductor

```
Lxx n1 n2 [l = lval] [model_name] [tc1 = val] [tc2 = val]
     + [scale = val] [ic = val] [m = mval] [r = val] [dtemp = val]
```

Description

Defines a linear inductor, where `n1` and `n2` are the terminals. Other HSPICE self inductor formats and instance parameters are not supported.

Arguments

`n1, n2`	Inductor terminal nodes
`l` = *`lval`*	Inductance of the inductor (in Henries)
`m = mval`	Multiplier to simulate parallel inductors (default = 1)
`tc1` = *`val`*	First-order temperature coefficient for the inductor
`tc2` = *`val`*	Second-order temperature coefficient for the inductor
`scale` = *`val`*	Scaling factors; scales inductance by its value (default = 1.0)
`ic` = *`val`*	Initial current through an inductor (this value is used as the DC operating point current when `uic` is specified in the `.tran` statement, and can be overwritten with an `.ic` statement)
`dtemp` = *`val`*	Temperature difference between the element and the circuit in Celsius (default = 0.0)
`r` = *`val`*	Resistance of the inductor in ohms (default = 0.0)

Examples

In the following example

```
L001 1 0 5e-6
```

defines an inductor named `L001` with an inductance of 5e-6 Henry connected between nodes `1` and `0`.

In the next example

```
L002 x y 1.5e-6 tc1 = 0.001 tc2 = 0
```

defines an inductor named `L002` with an inductance of 1.5e-6 Henry between nodes `x` and `y`. The temperature coefficients are 0.001 and 0.

In SPICE format:

```
l1 1 2 1n ic=1.0u
```

In Virtuoso Spectre format:

```
l1 1 2 inductor l=1n ic=1.0u
```

or

```
l1 1 2 inductor l=1n
ic l1:1=1.0u
```

Note: The Virtuoso UltraSim simulator provides hierarchical detection of inductor loops and generates an error message when an illegal inductor loop is detected. If you receive an inductor loop error message, remove the loop from the netlist file before running the circuit simulation again.

Voltage-Controlled Current Sources (G-Elements)

Voltage-Controlled Capacitor

```
Gxx n+ n- vccap pwl(1) in+ in- [delta = val] [scale = val]
     + [m = val] [tc1 = val] [tc2 = val] x1, y1, x2, y2 ... [ic = val]
```

Voltage-Controlled Current Source

Behavioral

```
Gxx n+ n- [vccs] cur = 'equation' [max = val] [min = val] [scale = val]
```

Linear

```
Gxx n+ n- [vccs] in+ in- transconductance [max = val] [min = val]
     + [m = val] [scale = val] [tc1 = val] [tc2 = val] [abs = 1] [ic = val]
```

Piece-Wise Linear

```
Gxx n+ n- [vccs] pwl(1) in+ in- [delta = val] [scale = val]
     + [m = val] [tc1 = val] [tc2 = val] x1, y1, x2, y2 ... [ic = val]
```

Polynomial

```
Gxx n+ n- [vccs] poly(ndim) in+ in- ... inndim+ inndim-
     + [tc1 = val] [tc2 = val] [scale = val] [max = val]
     + [min = val] [abs = 1] p0 [p1 ...] [ic = vals]
```

Delay Element

```
Gxx n+ n- [vccs] delay in+ in- td = val
     + [tc1 = val] [tc2 = val] [scale = val] [npdelay = val]
```

Voltage-Controlled Resistor

Linear

```
Gxx n+ n- vcr in+ in- transfactor [max = val] [min = val]
     + [m = val] [scale = val] [tc1 = val] [tc2 = val] [ic = val]
```

Piece-Wise Linear

```
Gxx n+ n- vcr pwl(1) in+ in- [delta = val] [scale = val]
     + [m = val] [tc1 = val] [tc2 = val] x1, y1, x2, y2 ... [ic = val]

Gxx n+ n- vcr npwl(1) in+ in- [delta = val] [scale = val]
     + [m = val] [tc1 = val] [tc2 = val] x1, y1, x2, y2 ... [ic = val]
```

```
Gxx n+ n- vcr ppwl(1) in+ in- [delta = val] [scale = val]
     + [m = val] [tc1 = val] [tc2 = val] x1, y1, x2, y2 ... [ic = val]
```

Polynomial

```
Gxx n+ n- vcr poly(ndim) in+ in- ... inndim+ inndim-
     + [tc1 = val] [tc2 = val] [scale = val] [max = val]
     + [min = val] [abs = 1] p0 [p1 ...] [ic = vals]
```

Description

Defines voltage-controlled current sources (VCCSs), voltage-controlled resistors (VCRs), and voltage-controlled capacitors (VCCAPs) in behavioral, linear, piece-wise linear, poly, and delay forms. In the behavioral function, the equation can contain terms of node voltages. In linear form, the output value is estimated with '[v(in+)-v(in-)]' multiplied by `transfactor` or `transconductance`, followed by the `scale` and temperature adjustment, before confined with the `abs`, `min`, and `max` parameters. In the piece-wise linear function, at least two pairs of voltage-current (or voltage-resistance, voltage-capacitance) points are required.

Arguments

`n+`, `n-`	Terminals of controlled element.
`in+`, `in-`	Positive and negative controlling nodes.
`vcr`, `vccap`, `vccs`	Keywords for the voltage-controlled resistor, capacitor, and current source elements. **Note:** `vcr`, `vccap`, and `vccs` are reserved words that cannot be used as node names.
`cur` = *'equation'*	Current of the controlled element flowing from `n+` to `n-`. It can be ■ An expression with parameters and functions of node voltages ■ Branch currents of other elements ■ Time, frequency, or temperature
`max` = *val*	Maximum value of the controlled current or resistance.
`min` = *val*	Minimum value of the controlled current or resistance.
`transconductance`	Voltage to current conversion factor.
`transfactor`	Voltage to resistance conversion factor.
`scale` = *val*	Scaling factor; scales current by its value (default = 1.0).

`m` = *`val`*	Multiplier (default = 1).
`tc1` = *`val`*	First-order temperature coefficient for the element.
`tc2` = *`val`*	Second-order temperature coefficient for the element.
`abs`	Output current takes its absolute value if `abs` = 1.
`ic` = *`val`*	Initial value of the current source (default = 0.0).
`delta` = *`val`*	A value used to smooth corners of the piece-wise linear function. The default is 1/4 of the smallest distance between break points, and is not to exceed 1/2 of this value.
`x1...`	Voltage drops between the controlling nodes `in+` and `in-`. They must be in ascending order.
`y1...`	Element output value corresponding to `x1...`
`npdelay`	The number of data points used in delay simulations.

The `npwl` and `ppwl` functions are used to interchange the `n+` and `n-` nodes, but use the same transfer function.

npwl

For the `in-` node connected to `n+`, if v(n+,n-) < 0, then the controlling voltage is v(in+,in-). Otherwise, the controlling voltage is v(in+,n-).

For the `in-` node connected to `n-`, if v(n+,n-) > 0, then the controlling voltage is v(in+,in-). Otherwise, the controlling voltage is v(in+,n+).

ppwl

For the `in-` node, connected to `n+`, if v(n+,n-) > 0, then the controlling voltage is v(in+,in-). Otherwise, the controlling voltage is v(in+,n-).

For the `in-` node, connected to `n-`, if v(n+,n-) < 0, then the controlling voltage is v(in+,in-). Otherwise, the controlling voltage is v(in+,n+).

Note: If the `in-` node does not connect to either `n+` or `n-`, the Virtuoso UltraSim simulator changes `npwl` and `ppwl` to `pwl`.

Examples

In the following example

```
G1 1 2 cur = '3.0*sin(v(7)/2)+v(6)^2'
```

defines a VCCS connected to nodes 1 and 2, with its current dependent on the voltage of nodes 6 and 7 in the given form.

In the next example

```
G2 1 2 vccs 5 0 0.5 max = 5 min = 0 m = 2 ic = 0
```

defines a VCCS connected to nodes 1 and 2. Its current is initialized as 0, and is half of the voltage at node 5. The current is also confined within 0 and 5 amps. The output current is multiplied by 2.

In the next example

```
G3 1 2 vccs pwl(1) 5 0 delta = 0.2 0, 0 0.5,1 1.5,1.5 scale = 1.e-3
```

defines a VCCS connected to nodes 1 and 2, its current controlled by the voltage at node 5. The current is calculated in a piece-wise linear function with a smoothing parameter of 0.2, and is scaled by 1.e-3 upon output.

In the next example

```
Gres 1 2 vcr pwl(1) 5 4 m = 3 0,0 1,1k 2,1.5k 3,1.8k 4,2.0k 5,2.0k ic = 1k
```

defines a VCR connected to nodes 1 and 2, with its resistance dependent on the voltage difference between nodes 5 and 4 in a piece-wise linear form. The initial resistance is 1k. The output resistance is decreased by 2/3.

In the next example

```
Gcap 1 2 vccap pwl(1) 5 4 m = 3 scale = 1.e-12 0,0 1,10 2,15 3,18 4,20 5,20 ic = 10
```

defines a VCCAP connected to nodes 1 and 2, with its capacitance dependent on the voltage difference between nodes 5 and 4 in a piece-wise linear form. The initial capacitance is set to 10 p after being scaled with 1e-12. The output capacitance is increased by a factor of 3.

In the next example

```
Gnmos d s vcr npwl(1) g s m =3 0,5g 1,5meg 2,5k 3,1k 5,50
```

tells the Virtuoso UltraSim simulator to model the source-drain resistor of the n-channel MOSFET which is used as a switch. Based on the npwl function, the resistor value (Gnmos) does not change when changing the position of the d and s nodes.

Voltage-Controlled Voltage Source (E-Elements)

Behavioral

```
Exx n+ n- [vcvs] vol = 'equation' [max = val] [min = val]
```

Linear

```
Exx n+ n- [vcvs] in+ in- gain [max = val]
     + [min = val] [scale = val]
     + [tc1 = val] [tc2 = val] [abs = 1] [ic = val]
```

Piece-Wise Linear

```
Exx n+ n- [vcvs] pwl(1) in+ in- [delta = val] [scale = val]
     + [tc1 = val] [tc2 = val] x1 y1 x2 y2 ... [ic = val]
```

Polynomial

```
Exx n+ n- [vcvs] poly(ndim) in+ in- ... inndim+ inndim-
     + [tc1 = val] [tc2 = val] [scale = val] [max = val]
     + [min = val] [abs = 1] p0 [p1 ...] [ic = vals]
```

Delay Element

```
Exx n+ n- [vcvs] delay in+ in- td = val
     + [tc1 = val] [tc2 = val] [scale = val] [npdelay = val]
```

Laplace

```
Exx n+ n- laplace in+ in- k0, k1, ..., kn/ d0, d1, ..., dm
     + [tc1 = val] [tc2 = val] [scale = val]
```

- Laplace Function

$$H(s) = \frac{k_0 + k_1 s + \ldots + k_n s^n}{d_0 + d_1 s + \ldots + d_m s^m} = \frac{0.0 + 0.0s + 0.0s^2 + 1.0s^3}{1.0 + 2.0s + 2.0s^2 + 3.0s^3}$$

Description

Defines six forms of voltage-controlled voltage sources (VCVSs): Behavioral, linear, piece-wise linear, polynomial, delay element, and Laplace. In behavioral form, the equation can contain terms of node voltages. In linear form, the output value is estimated using '`gain`*[v(in+)-v(in-)]', followed by the multiplication of `scale` and temperature adjustment, before being confined by the `abs`, `min`, and `max` parameters. In the piece-wise linear function, at least two pairs of voltage points are required.

Arguments

`n+, n-`	Terminals of the controlled element.
`in+, in-`	Positive and negative controlling nodes.
`vol` = *'equation'*	Voltage of the controlled element. The equation can be a function of parameters, node voltages, and branch currents of other elements.
`max` = *val*	Maximum value of the controlled voltage.
`min` = *val*	Minimum value of the controlled voltage.
`gain`	Voltage gain.
`scale` = *val*	Scaling factor; scales voltage by its value (default = 1.0).
`tc1` = *val*	First-order temperature coefficient for the element.
`tc2` = *val*	Second-order temperature coefficient for the element.
`abs`	Output voltage takes its absolute value if `abs` = 1.
`ic` = *val*	Initial value of the voltage source (default = 0.0).
`delta` = *val*	A value used to smooth the corners in the piece-wise linear function. It is defaulted to be 1/4 of the smallest distance between break points, not to exceed one-half of this value.
`x1...`	Voltage drops between the controlling nodes in+ and in-.
	They must be in ascending order.
`y1...`	Element voltages corresponding to x1...
`ndim`	Polynomial dimension (default = 1).
`p0, p1, ...`	Polynomial coefficients. If one coefficient is specified, it is assumed to be p1 (p0 = 0.0), representing a linear element. If more than one coefficient is specified, it represents a non-linear element.
`td`	Time delay keyword.
`npdelay`	Sets the number of data points to be used in delay simulations.
`k0, k1, ..., d0, d1, ...`	Laplace coefficients.

Examples

In the following example

```
E1 1 2 vol = '3.0*sin(v(7)/2)+v(6)^2'
```

defines a VCVS that is connected to nodes `1` and `2`, with its voltage dependent on nodes `6` and `7` in the given expression.

In the next example

```
E2 1 2 vcvs 5 0 0.5 max = 5 min = 0 ic = 0
```

defines a VCVS that is connected to nodes `1` and `2`. Its voltage is initialized to be 0, and is half of the voltage of node `5`. The final voltage is confined within 0 and 5 volts.

In the next example

```
E3 1 2 vcvs pwl(1) 5 0 delta = 0.2 0, 0 0.5,1 1.5,1.5
```

defines a VCVS that is connected to nodes `1` and `2`, with its voltage dependent on the voltage of node `5`. The voltage is calculated in a piece-wise linear function with a smoothing parameter `delta = 0.2`.

In the next example

```
E4 out 0 laplace in 0 0.0,0.0,0.0,1.0 / 1.0,2.0,2.0,3.0
```

defines a VCVS where the voltage `v(out,0)` is controlled by the voltage `v(in,0)` using the Laplace function.

Supported HSPICE Sources

Listed below are descriptions of the HSPICE DC and transient source functions that are supported by the Virtuoso UltraSim simulator for independent current and voltage sources.

- dc on page 99
- exp on page 100
- pwl on page 101
- pwlz on page 102
- pulse on page 103
- sin on page 104
- pattern on page 105

dc

`dc=`*`dc_voltage`*

or

`dc=`*`dc_current`*

Arguments

`dc=`*`dc_voltage`*	The value of the DC voltage source
`dc=`*`dc_current`*	The value of the DC current source

Example

```
V1 1 0 [dc=] 5
```

Declares a voltage source named `V1` with a DC voltage of 5 volts.

exp

```
exp [(] v1 v2 [td1 [tau1 [td2 [tau2]]]] [)]
```

Arguments

`v1`	Initial value of voltage or current in volts or amps
`v2`	Pulsed value of voltage or current in volts or amps
`td1`	Rise delay time in seconds (default = 0.0)
`td2`	Fall delay time in seconds (default = td1+tstep)
`tau1`	Rise time constant in seconds (default = `tstep`)
`tau2`	Fall time constant in seconds (default = `tstep`)

Example

```
I1 1 0 exp(-0.05m 0.05m 5n 25n 10n 20n)
```

Defines a current source named `i1` that connects to node `1` and ground with an exponential waveform, which has an initial current of -0.05 mA at t=0, and a final current of 0.05 mA. At t=5ns, the waveform rises exponentially from -0.05 mA to 0.05 mA with a time constant of 25 ns. At t=10 ns, it starts dropping to -0.05 mA again, with a time constant of 20 ns.

pwl

`pwl [(] t1 v1 [t2 v2 … tn vn] [r = repeat_time] [td = delay ] [)]`

or

`pl [(] v1 t1 [v2 t2 … vn tn] [r = repeat_time] [td = delay ] [)]`

Arguments

`t1 v1 t2 v2 ...`	Time and value pairs describing a piece-wise linear waveform. The value is amps or volts, depending on the type of source.
`r` = *`repeat_time`*	Repeats the waveform indefinitely starting from the `repeat_time` time point.
`td` = *`delay`*	The time in seconds to delay the start of the waveform.

Example

```
v001 1 0 pwl( 0 5 9n 5 10n 0 12n 0 13n 5 15n 5 r = 9n)
```

Defines a voltage source named `v001` that connects to node `1` and ground with a PWL waveform from 0 n to 15 n, continually repeating from 9 n to 15 n.

pwlz

`pwlz [(] *t1 v1* [*t2 v2 ...ti Z... tn vn*] [r = *repeat_time*] [td = *delay* ] [)]`

Description

This function resembles the PWL function, except that some voltage values can be replaced by a keyword `z`, which stands for the high-impedance state. In this state, the voltage source is disconnected from the time point (with keyword `z`) to the following time point (with non-`z` state).

Example

```
v002 1 0 pwlz ( 0 Z 9n 5v 10n 0 12n 0 13n Z 15n 5v )
```

Defines a voltage source that connects to node `1` and ground with a PWLZ waveform from 9 ns to 13 ns, and from 15 ns to the end of simulation.

pulse

`pulse [(] v1 v2 [td [tr [tf [pw [per]]]]] [)]`

Arguments

`v1`	The initial voltage in volts
`v2`	The second voltage (the waveform swings between `v1` and `v2`)
`td`	The time from the beginning of the transient to the first onset of the ramp (default = 0)
`tr`	The rise time of the pulse (default = tstep)
`tf`	The fall time of the pulse (default =tstep)
`pw`	The pulse width (default = tstop)
`per`	The period of the pulse (default = tstop)

Example

```
v001 1 0 pulse (0 5 0 1n 1n 5n 10n)
```

Defines a voltage source named `v001` that connects nodes `1` and `0`. The pulse waveform swings between 0 and 5 volts. The waveform has no initial delay, and has the rise and fall times as 1 ns. The total pulse width is 5 ns with a 10 ns period.

sin

`sin [() vo va [freq [td [theta [phase] ] ] ] ()]`

Arguments

`vo`	Voltage or current offset in volts or amps.
`va`	Voltage or current root mean square (RMS) amplitude in volts or amps.
`freq`	Source frequency in Hz (default = 1/tstop).
`td`	Time delay before beginning the sinusoidal variation in seconds (default = 0.0), response is 0 volts or amps until the delay value is reached, even with a non-zero DC voltage.
`theta`	Damping factor in units of 1/seconds (default = 0.0).
`phase`	Phase delay in units of degrees (default = 0.0).

Example

In the following example

```
i001 1 0 sin(0.01m 0.1m 1.0e8 5n 1.e7 90)
```

defines a current source named `i001` that connects to node `1` and ground with a `sin` waveform, and has an amplitude value of 0.1 mA, an offset of 0.01 mA, a 100 MHz frequency, a time delay of 5 ns, a damping factor of 1.e7, and a phase delay of 90 degrees.

Notes

- Voltage source loops in circuit designs can create simulation problems. The Virtuoso UltraSim simulator provides hierarchical detection of voltage source loops and generates an error message when an illegal voltage loop is detected.
- Only a DC voltage loop, with a total loop voltage of 0, does not generate an error message. If you receive a voltage loop error message, remove the loop from the netlist file before running the circuit simulation again.

pattern

```
pat [(] high low tdelay trise tfall tsample
     + bstring1 [rb=val] [r=val]
     + [bstring2 [rb=val] [r=val]] ... [)]
```

or

```
pat [(] high low tdelay trise tfall tsample
     + [component1 ... componentn] [rb=val] [r=val] [)]
```

Description

The `pattern` function defines a bit string (b-string) or a series of b-strings and consists of four states, `1`, `0`, `m`, and `z`, which represent the high, low, middle voltage or current, and high impedance states, respectively.

Arguments

`pat`	Keyword for a pattern time-varying source.
`high`	High voltage or current value of the pattern source in volts or amps.
`low`	Low voltage or current value of pattern source in volts or amps.
`tdelay`	Delay time in seconds from the beginning of the transient to the first ramp occurrence.
`trise`	Duration of the ramp-up in seconds.
`tfall`	Duration of the ramp-down in seconds.
`tsample`	Time spent at each `0`, `1`, `m`, or `z` pattern value in seconds.
`bstring1`	Defines a bit string consisting of `1`, `0`, `m`, or `z`. The first alphabetic character must be `b`. ■ **`1`** represents the high voltage or current value. ■ **`0`** is the low voltage or current value. ■ **`m`** represents the value which is equal to `0.5*(vhigh+vlow)`. ■ **`z`** represents the high impedance state (only for voltage source). **Note:** The b-string cannot contain parameters.

`component1 ... componentn`	Defines a series of b-strings. Each component is a b-string. `rb` and `r` can be used for each b-string. **Note:** Brackets `[]` must be used.
`rb=val`	Keyword to specify the starting bit or component to repeat. The number is counted from left to right. ■ The default is `rb=1` (source repeats from the most left-hand bit or component). ■ The value of `rb` must be an integer. ■ If the value is larger than the length of the b-string, or the total the number of components, an error is reported by the Virtuoso UltraSim simulator. ■ If the value is less than `1`, the simulator automatically sets it to the default value of `1`. **Note:** The value of `rb` cannot be a parameter.
`r=val`	Keyword to specify how many times to repeat the b-string or the components. ■ The default value is `r=0` (no repeat). ■ The value of `r` must be an integer. ■ If `r=-1`, then the repeating operation runs continuously. ■ If the value is less than `-1`, the simulator automatically sets it to the default value of `0`. **Note:** The value of `r` cannot be a parameter.

Examples

In the following example

```
v1 1 0 pat (5 0 0n 1n 1n 5n b01000 r=1 rb=2 bm10z)
```

tells the Virtuoso UltraSim simulator to define an independent pattern voltage source named `v1` with a first b-string `01000` that executes once and repeats once from the second bit `1`, and then the second b-string `m10z` executes once (that is, the whole bit pattern is `010001000m10z`). The high voltage `1` is 5 volts, low voltage `0` is 0 volts, middle voltage `m` is 2.5 volts, rise and fall times are both 1 ns, and each bit sample time is 5 ns.

In the next example

```
.param high = 1.5 low = 0 td=0 tr=10n tf=20n tsample=60n
V2 1 0 pat(high low td tr tf tsample
+ [b01000 r=1 rb=2 bm10z] RB=2 R=2)
```

tells the simulator to define an independent pattern voltage source named `v2` with a whole bit pattern of `01000 1000 m10z m10z m10z`.

Supported SPICE Format Simulation and Control Statements

Listed below are the Virtuoso UltraSim SPICE format simulation and control statements. The following sections provide a brief description of each statement.

.alter

```
.alter
```

Description

This statement is designed for repeating simulations under different conditions: Altered parameters, temperatures, models, circuit topology (different elements and subcircuit definitions), and analysis statements. Multiple `.alter` statements can be used in a netlist file, which is divided into several sections. The part before the first `.alter` statement is called the main block. Subsequent `.alter` statements and those between `.alter` and `.end` are referred to as alter blocks. When simulating an alter block, the information in the alter block is added to the main block, where conditions with identical names (for example, parameters, elements, subcircuits, and models) are replaced with those in the alter block. Analysis statements are treated in the same way.

The output from a sequence of altered simulations is distinguished by the number appended to the end of the filename, labeling the order in which they are generated. For example, the files from measures are named with `.mt0`, `.mt1`, and so forth. All the others skip the 0 for the first simulation, and appear as `.fsdb`, `.trn`, `.dsn`, `.nact`, `.pa`, `.ta`, `.vecerr`, and `.veclog`.

Examples

In the following example

```
x001 n1 n2 inv
...
.alter
x001 n1 n2 inv2
```

the call `x001` to the subcircuit `inv` from the main netlist file block is replaced by a call to subcircuit `inv2` in the altered simulation.

In the next example

```
.temp 25
.alter
.temp 50
```

the first simulation is run at 25 C and the second simulation is run at 50 C.

.connect

```
.connect node1 node2
```

Description

Use to connect node1 and node2.

Note: Both nodes must be at the same level of the design.

Arguments

`node1`, `node2`	Node names

Example

```
.connect vdd vdd!
```

Tells the simulator to connect the `vdd` and `vdd!` nodes. If probed, the nodes are retained in the waveform file.

.data

```
.data name param1 [param2 ...]
     + val11 [val21 ...]
     + val12 [val22 ...]
     .enddata
```

Description

This statement allows you to perform data-driven analysis in which parameter values can be modified in different simulations. This statement is used in conjunction with an analysis statement (for example, `.tran`) with a keyword `data` = name.

The Virtuoso UltraSim simulator only supports an inline format for the `.data` statement.

Arguments

`name`	Specifies the data name used in analysis statements
`param1, param2, ...`	Specifies the parameter names used in the netlist file (the names must be declared in a `.param` statement)
`val11, val21, ...`	Specifies the parameter values

Example

```
.param res = 1 cap = 1f
.tran 1ns 1us sweep data = allpars
.data allpars res cap
+ 1k 1p
+ 10k 10p
.enddata
```

Tells the Virtuoso UltraSim simulator to perform two separate simulations with the two pairs of parameters: `res` and `cap`.

.end

```
.end
```

Description

This statement specifies the end of the netlist file description. All subsequent statements are ignored.

Example

```
*title
...
.end
```

.endl

```
.endl
```

Description

This statement specifies the end of a library definition.

Example

```
.lib tt
...
.endl tt
```

.ends or .eom

```
.ends
```

or

```
.eom
```

Description

These statements specify the end of a subcircuit definition. Subcircuit references or calls can be nested within subcircuits.

Examples

```
.subckt inv in out
...
.ends
```

or

```
.macro inv in out
...
.eom
```

.global

```
.global node1 [node2 ... noden]
```

Description

This statement defines global nodes. The Virtuoso UltraSim simulator connects all references to a global node name, which can be used at any hierarchical level, to the same node.

Example

```
.global vdd gnd
```

Tells the Virtuoso UltraSim simulator to define nodes `vdd` and `gnd` as global nodes.

.ic

```
.ic v(node1) = val1 [v(node2) = val2] ... [v(noden) = valn] subckt=subckt_name
     depth=depth_val
```

Description

This statement is used to specify an initial voltage condition for nodes. The Virtuoso UltraSim simulator forces the node voltage to the specified voltage at time=0. A node name can be hierarchical and can contain wildcards. The statement can be embedded in the scope of a subcircuit. In this case, the initial condition is assigned to the nodes local to the subcircuit. If a conflict occurs between an embedded IC and a hierarchical IC, the embedded one is adopted.

For more information about wildcards, see "Wildcard Rules" on page 55.

Arguments

v(node)	Sets the initial voltage for the node. The node name can be hierarchical and can contain wildcards (for example, x?1.*.n*). In this case, the Virtuoso UltraSim simulator assigns the initial condition to all the nodes that match the name.
`subckt`	Specifies the subcircuit name (by default, applies to the top level). If the statement is already used in a subcircuit definition, this parameter is ignored. Setting the parameter is equivalent to defining the statement within a subcircuit declaration.
`depth`	Specifies the depth in the circuit hierarchy that a wildcard name applies to. This parameter is only available when the `*` wildcard is used in the output variable. If set to 1, only the nodes at the current level are applied (default value is infinity).

Example

```
.ic v(n1) = 0.5 v(n2) = 1.5 subckt=inv
```

Tells the Virtuoso UltraSim simulator to initialize node `n1` to 0.5 V and node `n2` to 1.5 V in all instances of subcircuit `inv`.

.include

`.include [filepath]`*`filename`*

Description

This statement inserts the contents of the file into the netlist file.

Note: `[filepath]`*`filename`* can be enclosed by single or double quotation marks.

Example

```
.include options.txt
```

Tells the Virtuoso UltraSim simulator to insert the `options.txt` file into the netlist file.

.lib

`.lib [libpath]` *`library_name section_name`*

Description

This statement is used to read common statements, such as device models, from a library file.

Arguments

`[libpath]`	Path to the library file
`library_name`	Name of the library file
	Note: The `[libpath]` *`library_name`* can be enclosed with single or double quotation marks.
`section_name`	Section of the library to be included

Example

```
.lib 'models.lib' tt
```

Tells the Virtuoso UltraSim simulator to read the `tt` section from the `models.lib` library file.

.nodeset

`.nodeset` *`v(node1)`* `=` *`val1`* `[v(node2) =` *`val2`*`] ... [v(`*`noden`*`) =` *`valn`*`]`

or

`.nodeset` *`node value`*

Description

This statement specifies the node set for the node. The Virtuoso UltraSim simulator forces the node voltage to the specified value at the first operating point iteration and then the solver calculates the node voltage used at time=0. A node name can be hierarchical and can contain wildcards. The statement can be embedded in the scope of a subcircuit. In this case, the initial condition is assigned to the nodes local to the subcircuit. If a conflict occurs between an embedded IC and a hierarchical IC, the embedded one is adopted.

For more information about wildcards, see "Wildcard Rules" on page 55.

Note: The `.nodeset` statement can be used to enhance convergence in DC analysis. If the node value is set close to the actual DC operating point, convergence can be enhanced.

Arguments

`v(node)`	Sets the node set for the node. The node name can be hierarchical and contain wildcards (for example, x?1.*.n*). In this case, the Virtuoso UltraSim simulator assigns the initial condition to all the nodes that match the name.

Example

```
.nodeset v(n1) = 0.5 v(n2) = 1.5
```

The initial starting point for the operating point calculation is 0.5 V for `n1` and 1.5 V for `n2`. The final operating point for both nodes may be slightly different since `.nodeset` is only used at the first iteration.

.op

```
.op [format] [time] [format] [time]
```

Description

The `.op` command is used to perform an operating point analysis. The Virtuoso UltraSim simulator reports all node voltages in an `.ic` file. If multiple time points are specified, the Virtuoso UltraSim simulator saves the node voltages in the following order: The first time point in an `.ic0` file and the second point in an `.ic1` file.

In addition, the Virtuoso UltraSim simulator reports all the operating point information in a different file (the file name is dependent on the keyword format used). If time is not specified, the Virtuoso UltraSim simulator performs the operating point analysis at time 0. If transient analysis is not available, the Virtuoso UltraSim simulator performs the operating point analysis at time 0, even if a non-zero time is specified with the command.

The Virtuoso UltraSim simulator can print the operating point analysis in the following formats: ASCII, PSF ASCII, and PSF binary (default is ASCII). To specify PSF ASCII, use `usim_opt wf_format=psfascii`. To specify PSF binary, use `usim_opt wf_format=psf`.

Arguments

`format` Specifies the report format and uses the following keywords: `all`, `current`, or `voltage`.

Note: Only one argument keyword can be used at a time in a command (you only need to use the first letter of the keyword).

- **Voltage** tells the Virtuoso UltraSim simulator to print a voltage table for each node and information for each model.

  The information is saved in an ASCII `voltage.op` file. If PSF format is specified, the file name is `tran_voltage_op.tran_op`.

- **All** tells the Virtuoso UltraSim simulator to print a voltage table for all the nodes, information for each model, and operating point information for each element (voltage, current, conductance, power, and capacitance).

  The information is saved in an ASCII `all.op` file. If PSF format is specified, the file name is `tran_all_op.tran_op`. Refer to the *Virtuoso UltraSim Waveform Interface Reference* for more details on PSF files.

- **Current** tells the Virtuoso UltraSim simulator to print a voltage table, information for each model, and limited operating point information for each element (voltage, current, and power).

  The information is saved in an ASCII `current.op` file. If PSF format is specified, the file name is `tran_current_op.tran_op`.

`time` Specifies the time at which the report is printed. This argument is placed directly after the `all`, `current`, and `voltage` arguments in the `.op` command.

Example

```
.op .5ns current 1.0ns 2.0ns
```

Tells the Virtuoso UltraSim simulator to calculate the operating point at 0.5 ns and prints the `.op` information in `voltage` format. The operating points are also calculated at 1.0 ns and 2.0 ns and printed in `current` format.

Note: If you use `.op [format] [time1] [time2]`, the format at `time2` is the same as `time1`.

.options

```
.options argument1 [argument2] ...
```

Description

This statement defines a set of SPICE options. The Virtuoso UltraSim simulator recognizes `dcap, defad, defas, defl`, `defnrd, defnrs, defpd, defps, defw, gmin, parhier, scale, scalm`, `search`, and `wl` as arguments to this statement.

Arguments

co = 80|132	Defines the number of variables to be displayed on each line (default value is `80`). If set to `80`, generates a narrow printout containing up to four output variables per line. If set to `132`, generates a wide printout containing up to eight output variables per line.
`dcap`	Equations calculate the depletion capacitance of the diodes (level=1|3) and BJTs.
`defad`	Default MOSFET drain diode area (`ad`). Default value is 0.
`defas`	Default MOSFET source diode area (`as`). Default value is 0.
`defnrd`	Default MOSFET drain resistor in number of squares (`nrd`). Default value is 0.
`defnrs`	Default MOSFET source resistor in number of squares (`nrs`). Default value is 0.
`defpd`	Default MOSFET drain diode perimeter (`pd`). Default value is 0.
`defps`	Default MOSFET source diode perimeter (`pd`). Default value is 0.
`defw`	Default MOSFET channel width. Default value is 1e-4.
`gmin` = *value*	The minimum conductance allowed for transient analysis. Default value is 1e-12.

`lngold = `*`0|1|2`*	Defines the numerical printout formats for the `.print` statement. ■ 0 - Engineering (default): 1.234K, 123M ■ 1 - G (fixed and exponential): 1.234e+03, .123 ■ 2 - E (exponential SPICE): 1.234e+03, .123e-1 **Note:** Engineering format cannot be combined with exponential format.
`parhier = `*`(local|global)`*	Rules for parameter passing; applies only to parameters with the same name, but under different levels of hierarchy. ■ **local** tells the simulator that a parameter name in a subcircuit overrides the same parameter name in a higher level of the hierarchy. ■ **global** tells the simulator that a parameter name at a higher level of the hierarchy overrides the same parameter name at a lower level.
`scale = `*`value`*	Scaling factor used to scale the parameters in the element card. Default value is 1.
`scalm = `*`value`*	Model scaling factor used to scale the model parameters defined in model cards. Default value is 1.
`search = `*`path`*	Specifies the search path for libraries and included files.
`wl = `*`(0|1)`*	`wl` changes the order of specified MOS elements from the default order length-width to width-length. Default value is 0.

.param

```
.param param1=value1 [param2 = val2 ... paramn = valn]
```

or

```
.param func_name='expression'
```

Description

This statement defines parameters and user-defined functions.

Examples

```
.param vcc=2.5
.param half_vcc='0.5*vcc'
.param g(x)='5*x+0.5'
.param f(x)='g(x)+5*x+0.5'
```

.subckt or .macro

```
.subckt subckt_name [port1 ... portn] [par1 = val1 ... parn = valn]
     [m = value]
```

or

```
.macro subckt_name [port1 ... portn] [par1 = val1 ... parn = valn] [m = value]
```

Description

These statements specify the beginning of a subcircuit definition. The subcircuit can have zero ports when all the nodes used in the subcircuit definition are declared global. A subcircuit definition can contain elements, subcircuit calls, nested subcircuit definitions, as well as simulation output statements (see "Supported SPICE Format Simulation Output Statements" on page 130). Parameters can be declared within subcircuit definitions, on a `.subckt` or `.macro` command, or on a subcircuit call. Multipliers are also supported on subcircuits (for example, m = 2).

Example

```
.subckt inv in out w = wval l = lval
m1 out in vdd vdd pmos w = vval*3 l = lval*2
m2 out in gnd gnd nmos w = wval l = lval
.eom
x1 n1 n2 inv w = 1e-06 l = 2.5e-07
```

Defines a subcircuit named `inv` that has two ports and takes two parameters, `w` and `l`. It is instantiated by a call named `x1`, which passes in values for `w` and `l`.

.temp

`.temp` *`val1`* `[`*`val2 ... valn`*`]`

Description

This statement defines the values of temperature used in the simulations.

Example

```
.temp 0 50 100
```

Tells the Virtuoso UltraSim simulator to perform simulations for three temperature values: `0`, `50`, and `100`.

.tran

```
.tran incr1 stop1 [incr2 stop2 ...incrn stopn] uic [start = value]
     + [sweep var type np start stop]
```

or

```
.tran incr1 stop1 [incr2 stop2 ...incrn stopn] uic [start = value]
     + [sweep var start=param_expr1 stop=param_expr2
     + step=param_expr3
```

or

```
.tran incr1 stop1 [incr2 stop2 ...incrn stopn] uic [start = value]
     [sweep data = dataname]
```

Description

This statement defines the transient analysis. In a transient analysis, the first calculation is a DC operating point using the DC equivalent model of a circuit. The DC operating point is then used as an initial estimate to solve the next time point in the transient analysis.

If `uic` is specified, the Virtuoso UltraSim simulator sets the node voltages as defined by `.ic` statements (or by the `ic = parameters` in various element statements) and sets unspecified nodes to 0 volts instead of solving the quiescent operating point. The DC operating points of unspecified nodes are set to 0 volts. In a SPICE netlist file, specifying `uic` has the same effect on the simulation as setting `usim_opt dc=0`.

Examples

In the following example

```
.tran 1e-12 1e-08 start = 0 sweep vcc lin 5 2.0 3.0
```

tells the Virtuoso UltraSim simulator to perform a transient analysis from 0 ns to 10 ns in steps of 1 ps. Additionally, `vcc` is swept linearly for five values from 2.0 to 3.0.

In the next example

```
.tran 1ns 1us sweep data = allpars
```

tells the simulator to perform a transient analysis from 0 ns to 1 us in steps of 1 ns. Additionally, the dataset `allpars` is used for performing the sweep.

In the next example

```
.tran 1ns 200ns uic
```

tells the simulator to perform a transient analysis from 0 ns to 200 ns in steps of 1 ns, without calculating the DC operating point when `uic` is used.

Supported SPICE Format Simulation Output Statements

Listed below are the supported Virtuoso UltraSim SPICE format simulation output statements. The following sections provide a brief description of each statement.

.lprobe and .lprint

```
.lprobe tran [low = value] [high = value] [name1 = ]ov1 [[name2 = ]ov2] ...
     [[namen = ]ovn] [depth = value] [subckt = name] [exclude = pn1]
     [exclude = pn2] ... [preserve=none|all|port]
```

```
.lprint tran [low = value] [high = value] [name1 = ]ov1 [[name2 = ]ov2] ...
     [[namen = ]ovn] [depth = value] [subckt = name] [exclude = pn1]
     [exclude = pn2] ... [preserve=none|all|port]
```

Description

These statements set up logic probes on nodes for the specified output quantity. The results are sent to a waveform output file. These statements can contain hierarchical names and wildcards for nodes or elements, and can be embedded within the scope of a subcircuit (for more information about wildcards, see "Wildcard Rules" on page 55).

The threshold voltages for `.lprint`/`.lprobe` can also be set using the `vl` and `vh` options. See "Threshold Voltages for Digital Signal Printing and Measurements" on page 220 for more information.

Note: Output variables can only be simple output variables.

Arguments

`tran`	Defines the analysis type (transient).
`ov1`, `ov2`	Specifies the simple output variables and uses *v(node_name)* format. The name can be hierarchical and contain wildcards (for example, x?1.*.n*).
`low` = *`value`*	Specifies the voltage threshold for the logic `0` (zero) state. The `0` (logic `low`) state is probed if the node voltage is less than or equal to `low`. If the node voltage is between `low` and `high`, the X state is probed. If not specified, the global parameter value `vl` is assigned.
`high` = *`value`*	Specifies the voltage threshold for the logic 1 (one) state. The `1` (logic `high`) state is probed if the node voltage is higher than or equal to `high`. If the node voltage is between `low` and `high`, the X state is probed. If not specified, the global parameter value `vh` is assigned.

`depth` = *`value`*	Specifies the depth in the circuit hierarchy that a wildcard name applies. If set as one, only the nodes at the current level are applied (default value is infinity).
`subckt = `*`name`*	Specifies the subcircuit this statement applies to. By default, it applies to the top level. If the statement is already in a subcircuit definition, this parameter is ignored. Setting this parameter is equivalent to defining the statement within a subcircuit declaration.
`exclude` = `pn1`, `pn2`	Specifies the output variables to be excluded from the probe. Names can be node or element names, and can contain wildcards.
`preserve=none|all|port`	Defines the content of nodes probed with wildcard probing. ■ **none** probes all nodes and ports connected to active devices (default). Nodes connected only to passive elements are not probed. ■ **all** probes all nodes, including nodes connected to passive elements, and probes all ports. ■ **port** only probes ports in subcircuits.

Examples

In the following example

```
.lprobe low = 0.5 high = 4.5 v(n1)
```

the voltage on node `n1` is converted to logic values using the `low` and `high` thresholds, and then output to the waveform output file.

In the next example

```
.lprobe low = 0.5 high = 4.5 v(*) v(BUF.n1) depth = 2 subckt = INV
```

the logic states are probed for all the nodes within the subcircuit named `INV` and one level below in the circuit hierarchy. In this case, the reported names of `BUF` are appended with the circuit call path from the top level to `INV`. This is equivalent to the situation where the statement '.lprobe tran v(*) depth = 2' is in the subcircuit definition of `INV` in the netlist file.

In the next example

```
.lprobe tran v(*) subckt=VCO preserve=all
```

RC reduction is constrained to preserve all nodes in `VCO`. Voltage probing is performed for all nodes in `VCO`, including internal nodes that are only connected to resistors and capacitors.

In the next example

```
.lprobe tran v(*) exclude=net* exclude=bl*
```

probes all node voltages except the voltages for nodes matching the pattern `net*` and `bl*`. The high and low threshold voltage is set by global parameters `vh` and `vl`, respectively.

In the next example

```
.lprobe low = 0.5 high = 4.5 v(*) exclude=*$*
```

or

```
.lprint low = 0.5 high = 4.5 v(*) exclude=*$*
```

the voltage on all nodes is converted to logic values using the low and high thresholds, and then output to the waveform output file. Nodes containing the `$` symbol are excluded.

.malias

```
.malias model_name=alias_name1 <alias_name2 ...>
```

Description

The Virtuoso UltraSim simulator supports `.malias`, an option used to create an alias name for a model. To create an alias, specify the following in the netlist file:

```
.malias model_name=alias_name1 <alias_name2 ... >
```

You can use `alias_name1 ...` the same way as the `model_name`.

Note: This option is only supported at the top level of the netlist file.

Arguments

`alias_name`	Specifies the alias name used for the model
`model_name`	Specifies the model name

.measure

```
.measure tran|tran_cont meas_name trig ... targ ...
```

Description

This statement defines the measurement that is performed for propagation, delay, rise time, fall time, average voltage, peak-to-peak voltage, and minimum and maximum voltage over a specified period, and over a number of other user-defined variables. The measurement can be used for power analysis on elements or subcircuits (see Examples).

The continuous measurement feature of the Virtuoso UltraSim simulator can be enabled by specifying the `tran_cont` option in the `.measure` statement. This type of measure performs the specified measurement continuously until the simulation ends. A measure output file named `cont_`*<meas_name>*`.mtx` is generated and reports the continuous measurement results.

The `.measure` statement can also be embedded within a subcircuit definition in the netlist file. The measure name is appended with the call path name from the top-level to the instances of the subcircuit. The `.measure` statement can also be used to perform the measurement of all output variables, including expression probes already defined in the `.probe expr()` statement.

The Virtuoso UltraSim simulator supports linked measure statements applicable for all the measure functions listed below. Some measure statements can depend on others by having names of other measures in expressions (instead of parameters). These expressions cannot contain node voltages and element currents. To avoid confusion, linked measure statements must be in the same scope (that is, either in the top level or in the same subcircuit definition).

Note: Any signal used in `.meas` is automatically saved in the waveform file.

The Virtuoso UltraSim simulator supports the following measure functions (descriptions include examples):

Average, RMS, Min, Max, Peak-to-Peak, and Integral

```
.measure tran meas_name func ov1 [from = value] [to = value]
```

Arguments

`tran`	Specifies the transient analysis for the measurement **Note:** The Virtuoso Ultrasim simulator only supports measurement of transient analysis.
`name`	User-defined measurement name
`ov1`	Name of the output variable (it can be the node voltage, branch current of the circuit, or an expression)
`func`	■ **avg** calculates the average area under *`ov1`*, divided by the period of time ■ **max** reports the maximum value of *`ov1`* over the specified interval ■ **min** reports the minimum value of *`ov1`* over the specified interval ■ **pp** reports the maximum value, minus the minimum of *`ov1`*, over the specified interval ■ **rms** calculates the square root of the area under the *`ov1`* curve, divided by the period of interest ■ **integ** reports the integral of *`ov1`* over the specified period
`from=`	Start time for the measurement period
`to=`	End time for the measurement period

Examples

The following example

```
.measure tran avg1 avg v(1) from = 0ns to = 1us
```

tells the Virtuoso UltraSim simulator to calculate the average voltage of node 1 from 0 ns to 1 us, evaluating the result with variable `avg1`.

In the next example

```
.measure tran Q2 integ I(out) from = 0ns to = 1us
```

tells the simulator to calculate the integral of `I(out)` from 0 ns to 1us, evaluating the result with variable `Q2`.

In the next example

```
.measure tran rms3 rms v(out) from = 0ns to = 1us
```

tells the simulator to calculate the RMS of the voltage on node `out` from 0 ns to 1 0ns, evaluating the result with variable `rms3`.

In the next example

```
.measure tran rout pp par('v(out)/i(out)')
```

tells the simulator to calculate the peak-to-peak value of the output resistance at node `out`, evaluating the result with variable `rout`.

Current and Power

Description

Used for current and power analysis on elements or subcircuits.

Examples

In the following example

```
.measure tran current max x0(xtop.x23.out) from=0ns to=1us
```

the maximum current of port `out` of instance `xtop.x23` is measured from 0 ns to 1 us, excluding all other lower hierarchical subcircuit ports.

In the next example

```
.measure tran power max `v(xtop.x23.out) * x0(xtop.x23.out)` from=0ns to=1us
```

the maximum power of port `out` of instance `xtop.x23` is measured 0 ns to 1 us, excluding all other lower hierarchical subcircuit ports.

In the next example

```
.measure tran current max x(xtop.x23.out) from=0ns to=1us
```

the maximum current of port `out` of instance `xtop.x23` and all instances below is measured.

In the next example

```
.measure tran power max `v(xtop.x23.out) * x(xtop.x23.out)` from=0ns to=1us
```

the maximum power of port `out` of instance `xtop.x23` and all instances below is measured.

In the next example

```
.measure tran power_avg avg `v(1) * i1(r1)` from=0ns to=1us
```

the average power on element `r1`, from 0 ns to 1 us, is measured in the circuit.

In the next example

```
.measure tran energy integ ` v(xtop.x23.out) * x(xtop.x23.out)` from=0ns to=10us
```

the integral power (total energy) of port `out` of instance `xtop.x23` and all instances below is measured.

Find and When

```
.measure tran meas_name find ov1 at = value
```

or

```
.measure tran|tran_cont meas_name find ov1 when ov2 = value [td = value]
     [rise = r|last | fall = f|last | cross = c|last]
```

or

```
.measure tran|tran_cont meas_name when ov1 = value|ov3 [td = value] [rise = r|last]
     [fall = f|last] [cross = c|last]
```

or

```
.measure tran|tran_cont meas_name when ov1 = ov2 [td = value] [rise = r|last] [fall
     = f|last] [cross = c|last]
```

Arguments

`tran`	Specifies the transient analysis for the measurement. **Note:** The Virtuoso Ultrasim simulator only supports measurement of transient analysis.
`meas_name`	User-defined measurement name.
`when|find`	Specifies the `when` and `find` functions.
`ov1, ov2, ov3`	Name of the output variable (it can be the node voltage, branch current of the circuit, or an expression).
`td`	Time at which measurement starts.

`rise=r`	Number of rising edges the target signal achieves `r` times (the measurement is executed).
`fall=f`	Number of falling edges the target signal achieves `f` times (the measurement is executed).
`cross=c`	Total number of rising and falling edges the target signal achieves `c` times (the measurement is executed). Crossing can be `rise` or `fall`.
`last`	Last `cross`, `fall`, or `rise` event (measurement is executed the last time the `find` or `when` condition is true). **Note:** `last` is a reserved keyword and cannot be used as a parameter name in `.measure` statements.
`from=`	Start time for the measurement period
`to=`	End time for the measurement period

Examples

```
.measure tran find1 find v(1) at = 0ns
.measure tran find2 find v(1) when v(2) = 2.5 rise = 1
.measure tran when1 v(1) = 2.5 cross = 1
.measure tran when2 v(1) = v(2) cross = 1
.measure tran_cont cont_find3 find v(1) when v(2) = 2.5 rise = 1
.measure tran_cont cont_when3 v(1) = 2.5 cross = 1
.measure tran_cont cont_when4 v(1) = v(2) cross = 1
```

Parameter

```
.measure tran meas_name param = 'expr'
```

Description

This format is specified together with other measures. `'expr'` can contain the names of other measures, but cannot contain node voltages or element currents.

Note: Since `'expr'` is a function of previous measurement results, it cannot be a function of node voltage or branch current.

Examples

In the following example

```
.measure tran avg1 avg v(1) from = 0ns to = 1us
.measure tran avg2 avg v(1) from = 2ns to = 3us
.measure tran avg12 param = ’avg1+avg2’
```

the measure `avg12` returns the sum of the values from `avg1` and `avg2`.

In the next example

```
.measure tran avg01 avg v(in) from = 0 to = 1e-08
.measure tran time1 when v(1) = 2.5 cross = 1
.measure tran delay1 trig at = 'time1' targ v(t4) val = '0.5*(avg01+0.0112)'
rise = 1
```

the measure `delay1` is calculated based on the results of `time1` and `avg`.

Rise, Fall, and Delay

```
. measure tran meas_name trig … targ …
```

Target

```
targ targ_var val = value [td = value] [cross = value | rise = value | fall = value]
```

Trigger

```
trig trig_var val = value [td = value] [cross = value] [rise = value] [fall = value]
```

or

```
trig at = value
```

Arguments

`tran`	Specifies the transient analysis for the measurement. **Note:** The Virtuoso Ultrasim simulator only supports measurement of transient analysis.
`meas_name`	User-defined measurement name.
`trig`	Specifies the beginning of trigger specifications.
`targ`	Specifies the beginning of target specifications.

`trig_var`	Name of the output variable that triggers the measurement. If the target is reached before the trigger activates, `.measure` reports a negative value.
`targ_var`	Name of the output variable the Virtuoso UltraSim simulator uses to determine the propagation delay with respect to `trig_var`.
`val=value`	Value of `trig_var` or `targ_var`.
`td=value`	Time the measurement starts. The simulator counts the number of `cross`, `rise`, or `fall` events that occur after the `td` value. Default=0.0.

`rise=value`
`fall=value`
`cross=value`
	Number of `rise`, `fall`, or `cross` events the target signal achieves `f` times (the measurement is executed).

`at=value`	Special case for trigger specification of measurement start time. The value can be a real time or a measurement result from a previous `.measure` statement.

Examples

In the first example

```
.measure tran delay1 trig v(1) val = 0.5 rise = 1 targ v(2) val = 0.5 fall =1
.measure tran_cont delay2 trig v(1) val = 0.5 rise =1 targ v(2) val = 0.5 fall =1
```

tells the Virtuoso UltraSim simulator to measure the delay from time point `v(1)`, when its value is 0.5 volts on the first rising edge, to time point `v(2)` when its values is 0.5 volts on the first falling edge.

The second `.measure` statement continuously reports the delay between v(1) and v(2) until the simulation ends. The additional output file is `cont_delay2.mt0`.

The next example

```
.measure tran delay1 trig at=1ns targ v(2) val = 0.5 fall = 1
```

tells the simulator to measure the delay from 1 ns to time point `v(2)` when its value is 0.5 volts on the first falling edge.

.probe, .print, .plot, and .graph

```
.probe [tran] [name1 = ]ov1 [[name2 = ]ov2] ... [[namen = ]ovn]
     [depth = value] [subckt = name] [exclude = pn1] [exclude = pn2] ...
     [preserve=all|port|none]

.print [tran] [name1 = ]ov1 [[name2 = ]ov2] ... [[namen = ]ovn]
     [depth = value] [subckt = name] [exclude = pn1] [exclude = pn2] ...
     [preserve=all|port|none]

.plot [tran] [name1 = ]ov1 [[name2 = ]ov2] ... [[namen = ]ovn]
     [depth = value] [subckt = name] [exclude = pn1] [exclude = pn2] ...
     [preserve=all|port|none]

.graph [tran] [name1 = ]ov1 [[name2 = ]ov2] ... [[namen = ]ovn]
     [depth = value] [subckt = name] [exclude = pn1] [exclude = pn2] ...
     [preserve=all|port|none]
```

Description

These statements are used to probe nodes and ports or to print simulation results in text and graphic formats. Using any of the specified keywords has the same effect, sending results to a waveform output file. The statements can contain hierarchical names and wildcards for nodes, ports, or elements, and can be embedded within the scope of a subcircuit.

For more information about wildcards, see "Wildcard Rules" on page 55.

For `.print`, the Virtuoso UltraSim simulator creates a *circuit.print#* output file for each simulation run (`#` starts at 0). The file includes all data for printed variables, with an `x` in the first column indicating where the `.print` output data starts, followed by a `y` indicating where the data ends.

The statements also support the following:

- Multiple statements in the netlist file
- Output variables in different formats (see Arguments on page 142)

Note: Nonexistent netlist file part names are ignored (warning message with names is printed).

Arguments

`tran`	Defines the analysis type (transient).

ov1, ov2 ... Name of the output variable (it can be a node voltage, branch current, port current, verilogA instance port voltage, or verilogA instance internal variable value).

- **v(node_name)** probes the *node_name* voltage. The *node_name* can be hierarchical, and can contain question marks and wildcards. For example: `v(x?1.*.n*)`.
- **i(element_name)** prints the branch current output through the element *element_name*. The *element_name* can be hierarchical, and can contain question marks and wildcards. For example: `i(x?1.*.n*)`
- **v1(element_name)** probes the voltage of the first terminal for the element `element_name`, v2 probes the voltage of the second terminal, v3 probes the voltage of the third terminal, and v4 probes the voltage of the fourth terminal (useful when the node name of a terminal is unknown; true for stitched devices).
- **x(instance_port_name)** returns the current flowing into the subcircuit port, including all lower hierarchical subcircuit ports. It can be used to probe power and ground ports of an instance, even if the ports are defined as a global node, and do not appear in the subcircuit port list. The *instance_port_name* can be hierarchical, and can contain question marks and wildcards. For example: `x(x?1.*.n*.vdd)`
- **x0(instance_port_name)** returns the current flowing into the subcircuit port, excluding all other lower hierarchical subcircuit ports. It can be used to probe power and ground ports of an instance, even if the ports are defined as a global node, and do not appear in the subcircuit port list. The *instance_port_name* can be hierarchical, and can contain question marks and wildcards. For example: `x0(x?1.*.n*.vdd)`.
- **vol = v(node1, node2)** probes the voltage difference between `node1` and `node2`, and assigns the result to the variable `vol`.

- **expr = par('expression')** probes the expression of simple output variables and assigns the result to `expr`. The expression can contain variables in the above two formats, as well as all the mathematical operators, and built-in or user-defined functions. An expression can also contain the names of other expressions.
- **var_name(veriloga_instance)** probes the `var_name` voltage for *`veriloga_instance`*. The `var_name` can be either a port name or an internal variable name of a verilogA module. The *`veriloga_instance`* is the instance name of a verilogA module, which can be hierarchical and can contain question marks and wildcards. For example: `PD(IO.AN?.B*)`.
- **all(veriloga_instance)** probes all the port voltages and internal variable values for *`veriloga_instance`*. The *`veriloga_instance`* can be hierarchical and can contain question marks and wildcards. For example: `all(IO.AN?.B*)`.

`depth=`*`value`* Specifies the depth in the circuit hierarchy that a wildcard name applies to. If it is set as one, only the nodes at the current level are applied (default value is infinity).

`subckt=`*`name`* Specifies the subcircuit this statement applies to. By default, it applies to the top level. If the statement is already in a subcircuit definition, this parameter is ignored. Setting this parameter is equivalent to defining the statement within a subcircuit declaration. Wildcards are supported.

`exclude=pn1, pn2` Specifies the output variables to be excluded from the probe. Names can be node or element names, and can contain wildcards.

`preserve= none|all|port` Defines the content of nodes probed with wildcard probing.

- **none** probes all nodes and ports connected to active devices (default). Nodes connected only to passive elements are not probed.
- **all** probes all nodes, including nodes connected to passive elements, and probes all ports.
- **port** only probes ports in subcircuits.

Note: To apply `.preserve=all` globally to all `.probe` statements in a netlist, set the `probe_preserve` option to `all` (see probe_preserve Option).

Examples

In the following example

```
.print v(n1) i1(m1) vdiff = v(n2,n3) expr1 = par('v(n1)+2*v(n2)')
```

tells the Virtuoso UltraSim simulator to print the voltage at node `n1` and the current `i1` for element `M1`. The voltage difference between nodes `n2` and `n3` is printed and assigned to `vdiff`. In addition, an expression of voltages at nodes `n1` and `n2` is printed and assigned to `expr1`.

In the next example

```
.print tran v(*) i(r1) depth = 2 subckt = VCO
```

tells the simulator to print the voltages for all nodes in the subcircuit named `VCO` and one level below in the circuit hierarchy. Also printed is the current of the resistor `r1` for all the instances of the subcircuit `VCO`. The reported names of `r1` are appended with the circuit call path from the top level to `VCO`. This is equivalent to the situation where the statement `.print tran v(*) i(r1) depth = 2` is written in the subcircuit definition of `VCO` in the netlist file.

In the next example

```
.print tran X(xtop1.block1.in) X0(xtop1.block1.in)
```

tells the simulator to report currents for instance `block1`, which is instantiated in top level block `xtop`. `X()` and returns the current into the subcircuit port `in`, including all lower hierarchical subcircuit ports. `X0()` only returns the current into the subcircuit port and excludes all other lower hierarchical subcircuit contributions.

In the next example

```
.print tran x0(xtop.x23.xinv.out)
```

tells the simulator to print the current of port `out` of instance `xtop.x23.xinv`, excluding all other lower hierarchical subcircuit ports.

Note: To print the subcircuit instance port current, use the format specified in this example.

In the next example

```
.print tran x(xtop.*)
```

tells the simulator to print the current of ports for instance `xtop` and all instances below.

In the next example

```
.probe tran v(*) subckt=VCO preserve=all
```

RC reduction is constrained to preserve all nodes in `VCO`. Voltages are probed for all nodes in `VCO`, including internal nodes that are only connected to resistors and capacitors.

In the next example

```
.probe tran v(*) exclude=net* exclude=bl* depth=2
```

probes all node voltages of the top level and one hierarchy below, except for the voltages of nodes matching the pattern `net*` and `bl*`.

In the next example

```
.probe tran v1(x1.x3.mp1) v2(x3.xp.mp4)
```

probes the drain of `x1.x3.mp1` and gate of `x3.xp.mp4`.

In the next example

```
.probe tran out(IO.ANA.VREG) ps3(IO.ANA.VREG) all(IO.ANA.C*)
```

probes the voltage of port `out` and the value of the internal variable `ps3` for the verilogA instance `IO.ANA.VREG`, as well as all the port voltages and internal variable values for verilogA instances that match the name `IO.ANA.C*`.

.print Control Options

`.options co=`*`80|132`*

This option is used to control printout width. The default value is `80`, with up to four variables per line in the printout file. If the number of variables in the `.print` statement exceeds four, then the first four variables are printed in the same line and the rest are printed on the next line. If `co = 132` is set, wide printout format is applied, allowing up to eight variables in a line.

`.width out=`*`80|132`*

Similar to the `co` option, `.width` is used to define the printout width of the output file (default is `80`). `out` is the keyword used for printout width.

The print time interval is determined by the `.tran` statement step time. If the netlist file contains Virtuoso Spectre® `.tran` options, then the `step` and `outputstart` arguments in the Spectre `.tran` option statement determine the print time step and the first print time point, respectively.

.option ingold = 0|1|2

The `ingold` option controls the numerical format of the printout (default value is `0`) and is specified with the `.option` statement. The engineering notation, in contrast to exponential format, provides two to three extra significant digits and aligns data columns to facilitate comparison.

`ingold=0`	Engineering format (default)	1.234K, 123M
`ingold=1`	G format (fixed and exponential)	1.234e+03, .123
`ingold=2`	E format (exponential SPICE)	1.234e+03, .123e-1

.option measdgt = x

The `measdgt` option formats the printed numbers in the `.measure` output files (such as `.meas0` and `.mt0`). `x` is used to specify the number of digits displayed to the right of the decimal point. The typical value of `x` is between 1 and 7 (default is 4). You can use `.option measdgt` with `.option ingold` to control the output data format.

Examples

Virtuoso UltraSim simulator format (preferred):

```
.option co=132
```

SPICE compatible format:

```
.width out=132
```

.option numdgt=x

The `numdgt` option formats the printed number in the `.print`, `.ic0`, and `voltage.op` output files. The `x` variable is used to specify the number of significant digits. The typical value of `x` is between 1 and 7 (default is 5). You can use `.option numdgt` and `.option ingold` to control the output data format. Using this option does not affect the accuracy of the simulation.

Examples

```
.option numdgt=7
```

Supported SPICE Format Expressions

The Virtuoso UltraSim simulator supports the following SPICE format expressions:

- Built-in functions
- Constants
- Operators

Built-In Functions

The Virtuoso UltraSim simulator supports the following built-in functions for Virtuoso Spectre and SPICE modes.

Table 2-2 Built-In Functions

Form	Function	Class	Description
sin(x)	sine	trig	Returns the sine of x in radians
cos(x)	cosine	trig	Returns the cosine of x in radians
tan(x)	tangent	trig	Returns the tangent of x in radians
asin(x)	arc sine	trig	Returns the inverse sine of x in radians
acos(x)	arc cosine	trig	Returns the inverse cosine of x radians
atan(x)	arc tangent	trig	Returns the inverse tangent of x in radians
sinh(x)	hyperbolic sine	trig	Returns the hyperbolic sine of x in radians
cosh(x)	hyperbolic cosine	trig	Returns the hyperbolic cosine of x in radians
tanh(x)	hyperbolic tangent	trig	Returns the hyperbolic tangent of x in radians
asinh(x)	hyperbolic inverse sine	trig	Returns the hyperbolic inverse sine of x in radians
acosh(x)	hyperbolic inverse cosine	trig	Returns the hyperbolic inverse cosine of x in radians
atanh(x)	hyperbolic inverse tangent	trig	Returns the hyperbolic inverse tangent of x in radians

Table 2-2 Built-In Functions, *continued*

Form	Function	Class	Description
atan2(x,y))	tangent inverse	trig	Returns the inverse tangent of x/y in radians
hypot(x,y)	hypotenuse	trig	Returns the square root of (x*x + y*y)
ln(x)	natural log	trig	Returns the natural log with base e of x
abs(x)	absolute value	math	Returns the absolute value of x: |x|
sqrt(x)	square root	math	Returns the square root of the absolute value of x: sqrt(-x) = -sqrt(|x|)
pow(x,y)	absolute power	math	Returns the value of x raised to the integer part of y: $x^{(\text{integer part of y})}$
pwr(x,y)	signed power	math	Returns the absolute value of x, raised to the y power, with the sign of x: (sign of x)$\|x\|^{y}$
log(x)	natural logarithm	math	Returns the natural logarithm of the absolute value of x, with the sign of x: (sign of x)log(|x|)
log10(x)	base 10 logarithm	math	Returns the base 10 logarithm of the absolute value of x, with the sign of x: (sign of x)$\log_{10}$(|x|)
exp(x)	exponential	math	Returns e, raised to the power x: e^{x}
db(x)	decibels	math	Returns the base 10 logarithm of the absolute value of x, multiplied by 20, with the sign of x:(sign of x)20$\log_{10}$(|x|)
int(x)	integer	math	Returns the integer portion of x
sgn(x)	return sign	math	Returns -1 if x is less than 0
			Returns 0 if x is equal to 0
			Returns 1 if x is greater than 0
sign(x,y)	transfer sign	math	Returns the absolute value of x, with the sign of y:(sign of y)|x|
gauss	Gaussian distribution function using relative variation	math	Returns only the nominal value

Table 2-2 Built-In Functions, *continued*

Form	Function	Class	Description
agauss	Gaussian distribution function using absolute variation	math	Returns only the nominal value
unif	uniform distribution function using relative variation	math	Returns only the nominal value
aunif	uniform distribution function using absolute variation	math	Returns only the nominal value
limit	limit distribution function using absolute variation	math	Returns only the nominal value
min(x,y)	smaller of two args	control	Returns the numeric minimum of x and y
max(x,y)	larger of two args	control	Returns the numeric maximum of x and y
ceil(x)	ceiling	algebraic	Returns the ceiling of x
floor(x)	floor	algebraic	Returns the floor of x
fmod(x,y)	fractional mod	algebraic	Returns the mod of x, y

Constants

The Virtuoso UltraSim simulator supports the following constants.

Note: Constants are only valid in Virtuoso Spectre mode.

Table 2-3 Constants

M_E : 2.7182818284590452354
M_LOG2E : 1.4426950408889634074
M_LOG10E : 0.43429448190325182765
M_LN2 : 0.69314718055994530942
M_LN10 : 2.30258509299404568402
M_PI : 3.14159265358979323846

Table 2-3 Constants, *continued*

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_SQRT2 : 1.41421356237309504880
M_SQRT1_2 : 0.70710678118654752440
M_DEGPERRAD : 57.2957795130823208772
P_Q : 1.6021918e-19
P_C : 2.997924562e+8
P_K : 1.3806226e-23
P_H : 6.6260755e-34
P_EPS0 : 8.85418792394420013968e-12
P_U0 : 0.000001256637061436
P_CELSIUS0 : 273.15

Operators

The Virtuoso UltraSim simulator supports the following operators.

Table 2-4 Operators

Form	Function	Description
+	add	Used for addition
-	subtract	Used for subtraction
/	divide	Used for division
*	multiply	Used for multiplication
**	power	Used for power
%	modulus	Used for modulus
<	less than (relational)	Returns 1 if the left operand is less than the right operand (otherwise returns 0)
>	greater than (relational)	Returns 1 if the left operand is greater than the right operand (otherwise returns 0)
<=	less than or equal (relational)	Returns 1 if the left operand is less than or equal to the right operand (otherwise returns 0)
>=	greater than or equal (relational)	Returns 1 if the left operand is greater than or equal to the right operand (otherwise returns 0)
!=	inequality	Returns 1 if the operands are not equal (otherwise returns 0)
==	equality	Returns 1 if the operands are equal (otherwise returns 0)
&&	logical AND	Returns 1 if neither operand is zero (otherwise returns 0)
||	logical OR	Returns 1 if either or both operands are not zero (returns 0 only if both operands are zero)
&	bitwise AND	Returns signed bitwise AND
|	bitwise OR	Returns signed bitwise OR

Table 2-4 Operators, *continued*

Form	Function	Description
cond?expr1:expr2	ternary	If cond is true, evaluates expr1 (if false, evaluates expr2)

3

Simulation Options

This chapter describes the simulation options that can be used to set the Virtuoso® UltraSim™ simulator for speed, accuracy, and functionality.

See the following topics for more information.

Setting Virtuoso UltraSim Simulator Options

The Virtuoso UltraSim simulator supports Virtuoso Spectre® and HSPICE (registered trademark of Synopsys, Inc.) netlist file formats. The Virtuoso UltraSim simulator options can be set in a Virtuoso Spectre netlist file using the `usim_opt` command, whereas the simulator options in a HSPICE netlist file require the `.usim_opt` command.

Spectre Syntax

```
usim_opt [opt1] [opt2] ... [scope1] [scope2] ...
```

SPICE Syntax

```
.usim_opt [opt1] [opt2] ... [scope1] [scope2] ...
```

You can set any number of Virtuoso UltraSim simulator options on the same `usim_opt` command line and also list the options in any order. These options can be set locally by using the `scope` option or globally (no scope).

The following scopes are supported by the Virtuoso UltraSim simulator:

- **Subcircuit instances:** `inst=[inst1 inst2 ...]`
- **Subcircuit primitives:** `subckt=[subck1 subckt2 ...]`
- **Subcircuit instance inside a subcircuit:** `subcktinst=[subckt1.xinst1 subckt2.xinst2 ...]`
- **Device model primitives:** `model=[model1 model2 ...]`
- **Model primitives inside a subcircuit:** `subcktmodel=[subckt1.model1 subckt2.model2 ...]`
- **Power network:** `scope=power`
- **Stitched network:** `scope=stitch`

Wildcards (`*`,`?`) can be used to match multiple scopes simultaneously (for more information about wildcards, see “Wildcard Rules” on page 55).

Note: If the scope includes multiple entries, or contains wildcards, it must be enclosed by `[]` brackets.

Example

Spectre Syntax:

```
usim_opt sim_mode=ms speed=6 postl=2
usim_opt sim_mode=a inst=i1.i2.vco1
usim_opt sim_mode=df subckt=[digital1 digital2]
```

HSPICE Syntax:

```
.usim_opt sim_mode=ms speed=6 postl=2
.usim_opt sim_mode=a inst=x1.x2.vco1
.usim_opt sim_mode=df subckt=[digital1 digital2]
```

The options of parent subcircuits are automatically inherited by child subcircuits called by the parent. If a local option is set for a child, it overrides the options inherited from the parent. When combining local and global options, the following rules apply:

- A lower level option overrides a higher level option
- Options do not need to be listed in a specific order in the netlist file

- An instance-based option overwrites subcircuit settings if applied to the same block
- The last option set overwrites a previously set option if applied to the same block

The Virtuoso UltraSim simulator also supports a common options configuration file called `ultrasim.cfg`, which enables you to set the simulator default options. This configuration file can be used to set netlist file, user, or site-specific Virtuoso UltraSim simulator options. Both the Virtuoso Spectre and HSPICE syntax options are supported in the configuration file. If the option defined in `ultrasim.cfg` is also defined in the netlist file, the netlist file overwrites the option. See "Virtuoso UltraSim Simulator Configuration File" on page 40 for more information about configuration files.

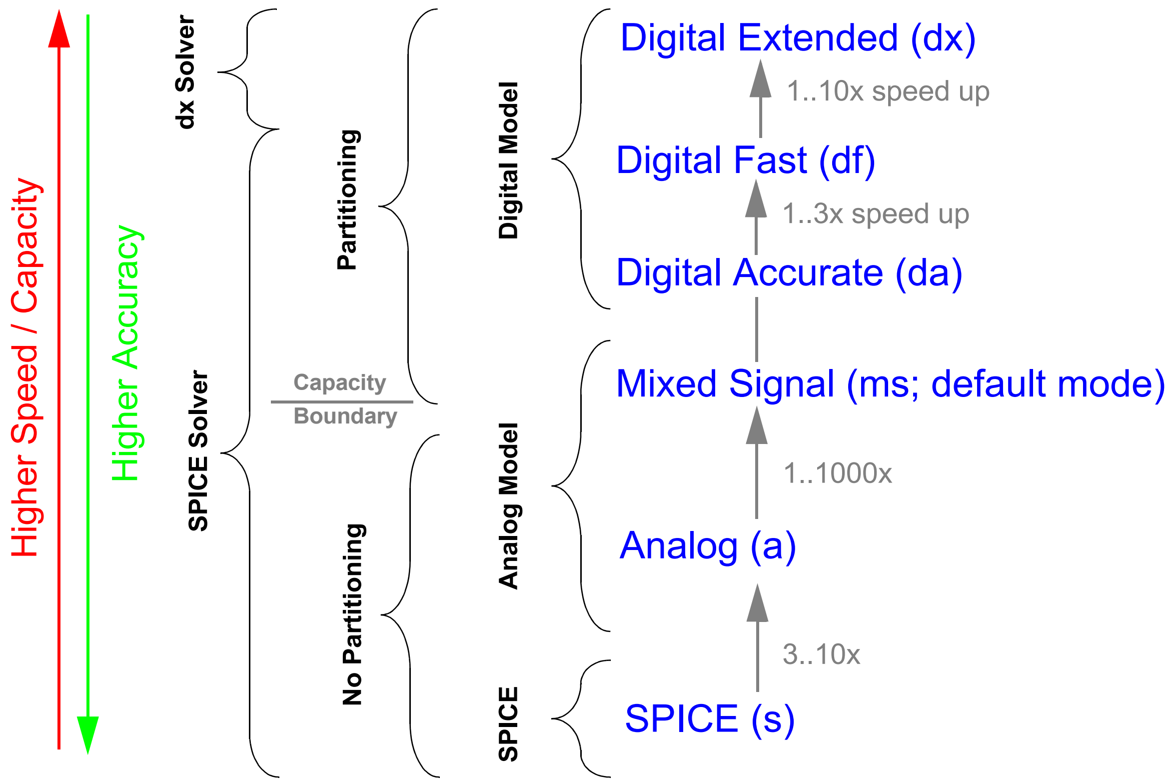
Simulation Modes and Accuracy Settings

You trade-off speed and accuracy by choosing between different model and simulation abstraction levels, and by adjusting the tolerances used by the Virtuoso UltraSim simulation algorithm. The simulation mode `sim_mode` determines the type of partitioning and device models the Virtuoso UltraSim simulator applies to the circuit. The available modes are digital extended (`dx`), digital fast (`df`), digital accurate (`da`), mixed signal (`ms`), analog (`a`), and SPICE (`s`). Within each simulation mode, the `speed` option specifies the accuracy and determines the relative tolerance used for voltage and current calculations (valid settings are 1 to 8).

Simulation Modes

Figure 3-1 on page 158 shows how simulation modes influence partitioning and device modeling. All simulation modes use the same SPICE solver. The `a` and `s` modes do not use partitioning, and the `ms`, `da`, `df`, and `dx` modes use more aggressive partitioning. In addition, `s` mode uses SPICE models, `a` and `ms` modes use analog representative models, and `df`, `da`, and `dx` modes use digital representative models.

Figure 3-1 Simulation Modes



- **Digital Extended (dx) mode** targets an accuracy of within 20% compared to `s` mode and is designed only for the functional verification of digital circuits. This is achieved by using a digital nonlinear current model, a constant capacitance model, and diffusion junctions with the metal oxide semiconductor field-effect transistor (MOSFET), as well as a special `dx` solver.

  Note: This mode is not applicable to memory or mixed signal design blocks, and may cause slow simulation speed, accuracy issues, or memory problems if used to simulate these types of design blocks.

- **Digital Fast (`df`) mode** targets an accuracy of within 10% compared to `s` mode and is designed for the functional verification of digital circuits and memories. This is achieved by using a digital nonlinear current model for the MOSFET, and a constant capacitance model for the MOSFET, and the MOSFET diffusion junctions. A partitioning algorithm is used to provide high-speed simulation.

- **Digital Accurate (`da`) mode** is used for timing verification of digital circuits and memories, and for some PLL and mixed signal designs. `da` mode employs a digital nonlinear current and charge model for the MOSFET and its diffusion junctions. `da` mode uses partitioning and targets a simulation error of less that 5%.

- **Mixed Signal Mode (`ms`) mode** provides the accuracy needed for analog, mixed signal, and PLL applications. It uses partitioning and an analog representative model for the MOSFET current and charge and diffusion junction. `ms` mode targets an accuracy within 3%.
- **Analog (`a`) mode** is designed for high-accuracy applications like ADC, DAC, and DC/DC circuits. It uses the same analog representative models as `ms` mode. It simulates the design in one partition, but provides a speed improvement of three to ten times over conventional SPICE simulation due to the analog representative model.
- **SPICE (`s`) mode** uses Berkeley SPICE models and is targeted to match other SPICE simulators (target error of 1%).

Table 3-1 on page 159 gives an overview of the Virtuoso UltraSim simulation modes, shows how they are related to device modeling, and tolerances within the simulation tool, and provides a basic understanding of what mode needs to be used for which application.

Table 3-1 Virtuoso UltraSim Simulation Modes Overview

Simulation Mode	`dx`	`df`	`da`	`ms`	`a`	`s`	**Option**
MOSFET	**Digital Model**			**Analog Model**		**SPICE**	
Current/Charge Model	`df`		`da`	`a`		`s`	mos_method
Differential Junction	`df`		`da`	`a`		-	mosd_method
JUNCAP	`a`					s	diode_method
Diode	`s`						
BJT	`s`						
JFET/MESFET	`s`						
Speed	1-8						speed
Default Speed (Tolerance)	8 (0.07)	5 (0.01)					speed (tol)
Integration Method	be				gear2		method
Partitioning	Digital				None		
Target Error	< 20%	< 10%	< 5%	< 3%	< 1%	< 1%	

Table 3-1 Virtuoso UltraSim Simulation Modes Overview

Simulation Mode	dx	df	da	ms	a	s	Option
Application	Functional verification of digital circuits only	Functional verification of digital circuits/ memories	Timing verification of digital circuits and memories, some mixed signal (MS) designs	MS and special memory designs	Analog and high sensitivity designs		

Supported Representative Models Summary

The following is a summary of MOSFET and diode models supported by the analog and digital representative models.

MOSFET

- BSIM3, BSIM4, MOS9, and MOS11 are supported by the analog and digital representative models
  - HSPICE and Spectre syntax is supported
- BSIMSOI (versions 2.23 and higher) and ssimsoi are supported by the analog and digital representative models
  - HSPICE and Spectre syntax is supported
- HiSIM2 is supported by the analog and digital representative models
  - HSPICE and Spectre syntax is supported

Diode

- juncap is supported by the analog representative model
  - HSPICE and Spectre syntax is supported
  - The analog representative model is the default for all simulation modes, except for `s` mode which uses the SPICE model

sim_mode

Description

Specifies the simulation mode that defines the partitioning and device model approach the Virtuoso UltraSim simulator applies to the circuit. Refer to Figure 3-1 on page 158 for more information.

Table 3-2 sim_mode Options

Option	Description
`sim_mode=dx`	Digital extended mode
`sim_mode=df`	Digital fast mode
`sim_mode=da`	Digital accurate mode
`sim_mode=ms`	Mixed signal mode (default)
`sim_mode=a`	Analog mode
`sim_mode=s`	SPICE mode

Caution

The digital extended (dx) mode only applies to digital designs, not memory circuits.

The `ms`, `da`, `df`, and `dx` modes use circuit partitioning, based on ideal power supplies (dc or pwl voltage source), and apply it only to the MOSFET portion of the design. Simulation performance may be degraded as a result of using:

- Generator circuits instead of ideal power supplies

  Solution: Use voltage regulator (VR) simulation (see Chapter 5, “Voltage Regulator Simulation” for more information).

- Other sources such as controlled, sinus, or current sources

  Solution: Use voltage regulator (VR) simulation (see Chapter 5, “Voltage Regulator Simulation” for more information).

- Post-layout resistors in the power supply

  Solution: Use rvshort or the power network solver (see Chapter 6, “Power Network Solver” for more information).

- Inductors in the power supply

  Solution: Use `a` mode to consider inductor behavior for the circuit or use lvshort to improve performance in `ms`, `da`, and `df` modes.

The bipolar junction transistor (BJT) or behavioral Verilog-A dominated designs cannot take advantage of circuit partitioning and other Fast SPICE technology. For these designs, Cadence recommends using `s` mode. For BiCMOS and latchup BJT designs, `ms` mode can provide a significant improvement in simulation performance.

Example

Spectre Syntax:

```
usim_opt sim_mode=da inst=xi1.xi5.xi3
```

SPICE Syntax:

```
.usim_opt sim_mode=da inst=xi1.xi5.xi3
```

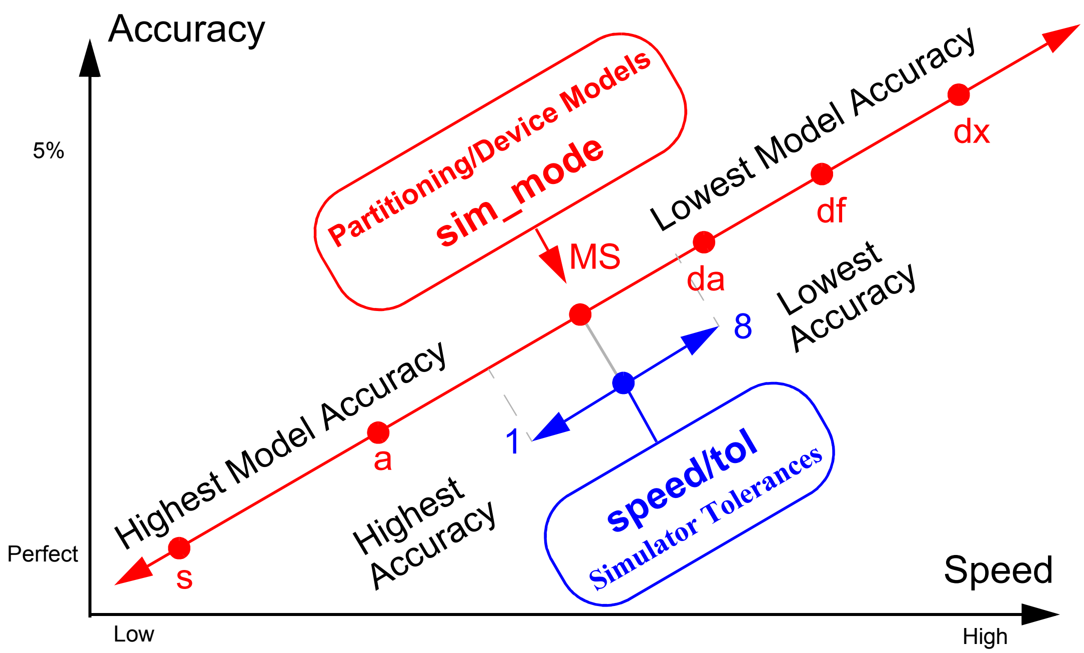
tells the simulator that subcircuit `xi1.xi5.xi3` and its children are simulated in `da` mode, but everything else is in `df` mode.

Accuracy Settings

The Virtuoso UltraSim simulator uses relative and absolute error tolerances while performing transient simulation. To simplify usage, it provides the high-level accuracy option `speed`, which allows you to customize the simulation speed and accuracy within each simulation mode. This option determines the relative convergence criterion (`tol`) for the current and voltage calculation.

Figure 3-2 on page 163 shows how simulation speed can be set for each simulation mode to trade-off speed and accuracy.

Figure 3-2 Accuracy Settings



speed

Description

Defines the simulation speed and accuracy within the chosen simulation mode, and determines the relative tolerance for voltage and current calculations. The default value is speed=5 for all simulation modes except `dx` mode (default is speed=8).

Table 3-3 speed Options

speed	tol	Mixed Signal (PLL/VCO)	Analog (ADC, SD)	Memory	Digital
1	0.0001	-	-	-	-
2	0.001	Applicable	Applicable	-	-
3	0.0025	Applicable	Applicable	-	-
4	0.005	Applicable	Applicable	Applicable	-
5	0.01	Applicable	Applicable	Applicable	Applicable
6	0.02	-	-	Applicable	Applicable

Table 3-3 speed Options, *continued*

speed	tol	Mixed Signal (PLL/VCO)	Analog (ADC, SD)	Memory	Digital
7	0.04	-	-	-	Applicable
8	0.07	-	-	-	Applicable

Note: Cadence does not recommend using `speed=7` /`8` together with `a` or `ms` mode because the speed settings may not provide sufficient simulation accuracy with these modes.

Example

Spectre Syntax:

```
usim_opt speed=1
```

SPICE Syntax:

```
.usim_opt speed=1
```

tells the Virtuoso UltraSim simulator to use a relative tolerance of 0.0001 to achieve high-accuracy results.

Recommended Simulation Modes and Accuracy Settings

Table 3-4 on page 164 provides option setting recommendations for different circuit types. Cadence suggests that you start with the options in column two of the table. Column three suggests how to achieve better accuracy and speed if the recommended options do not fulfill your requirements. See “Setting Virtuoso UltraSim Simulator Options” on page 155 for more information.

Table 3-4 Recommended Virtuoso UltraSim Simulation Options

Circuit Type	Option	Accuracy and Speed Adjustments
Digital Circuit	`sim_mode=df`	■ Adjust `speed` to trade off speed and accuracy
	`speed`=7	■ Use `sim_mode=ms`, `speed`=5 to simulate power consumption

Table 3-4 Recommended Virtuoso UltraSim Simulation Options, *continued*

Circuit Type	Option	Accuracy and Speed Adjustments
Static Random Access Memory (SRAM)	`sim_mode=df` `speed`=7	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=ms`, `speed`=5 to simulate power consumption
Dynamic Random Access Memory (DRAM)	`sim_mode=ms` `speed`=5	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=df` | `da` to improve speed
Flash Memory and EEPROM	`sim_mode=ms` `speed`=6	■ Adjust `speed` to trade off speed and accuracy ■ Apply `sim_mode=df` locally to large digital blocks ■ Apply `sim_mode=a` locally to oscillators and charge pumps to improve speed ■ Apply `mos_method=s` locally to memory cells
Read-Only Memory (ROM)	`sim_mode=df` `speed`=6	■ Adjust `speed` to trade off speed and accuracy ■ Apply `sim_mode=ms` locally to sensing path to improve accuracy
Phase-Locked Loop (PLL) and Delay-Locked Loop (DLL)	`sim_mode=ms` `analog`=2	■ Use `method=trap` | `gear2` for speed improvement ■ Apply `analog`=2 globally or `sim_mode=a` locally to VCO to achieve better accuracy and stability ■ Apply `sim_mode=df` locally to large digital dividers ■ Adjust `speed` to trade off speed and accuracy

Table 3-4 Recommended Virtuoso UltraSim Simulation Options, *continued*

Circuit Type	Option	Accuracy and Speed Adjustments
Voltage Controlled Oscillator (VCO) and Oscillator	`sim_mode=ms` `speed`=5 | 4	■ Use `method=trap` | `gear2only` to maintain oscillation ■ Apply `sim_mode=a` to achieve better accuracy and stability ■ Adjust `speed` to trade off speed and accuracy **Note:** May need to set `maxstep_window` or initial conditions to start oscillation.
Switch Capacitor (SC) Filter	`sim_mode=ms` `analog`=2 | 4	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=a` to improve accuracy
Analog Front End (AFE)	`sim_mode=ms` `analog`=2	■ Adjust `speed` to trade off speed and accuracy ■ Apply `sim_mode=a` to highly sensitive analog blocks, such as op amp, filter, and analog multiplier to improve accuracy
Sigma Delta Converter	`sim_mode=ms` `analog`=2 | 4	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=a` to improve accuracy
Operational Amplifier	`sim_mode=a`	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=s` to improve accuracy
Bandgap Reference	`sim_mode=ms`	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=a` | `s` to improve accuracy
Charge Pump and Switching Power Supply	`sim_mode=ms` `analog`=2 | 4	■ Adjust `speed` to trade off speed and accuracy ■ Use `sim_mode=a` to improve speed and accuracy
Power Management Circuit	`sim_mode=ms`	■ Adjust `speed` to trade off speed and accuracy ■ Apply `sim_mode=a` locally to sensitive analog blocks to improve accuracy
BICMOS Design	`sim_mode=ms`	■ Adjust `speed` to trade off speed and accuracy ■ Consider using `sim_mode=a` for smaller designs

Table 3-4 Recommended Virtuoso UltraSim Simulation Options, *continued*

Circuit Type	Option	Accuracy and Speed Adjustments
SOI SRAM	`sim_mode=ms`	■ Adjust `speed` to trade off speed and accuracy ■ Set `mos_method=s` globally to enhance convergence

Circuit with MOSFETs operating in weak inversion	`sim_mode=s`
	or
	`mos_method=s`
		■ Adjust `speed` to trade off speed and accuracy

ADC, DAC	`sim_mode=ms` `analog`=2 | 4	■ Adjust `speed` to trade off speed and accuracy ■ Apply `sim_mode=a` locally to sensitive analog blocks to improve accuracy ■ Apply `sim_mode=df` locally to large digital blocks to improve speed

RF Design (LNA, RFVCO, Mixer, and PA)	`sim_mode=ms`
	Local RF block options:
	`sim_mod =a`
	`speed`=3 | 4
		■ Adjust `speed` to trade off speed and accuracy
		■ Set `method=trap` | `gear2only` for VCO

High-Sensitivity Analog Option

The Virtuoso UltraSim simulator uses circuit partitioning in higher simulation modes to improve simulation performance. Setting the `analog` option allows you to select between aggressive, moderate, and more conservative partitioning. The `analog` option applies only to the `ms`, `da`, and `df` modes, since the `a` and `s` modes do not use partitioning.

analog

Description

Controls circuit partitioning, once you have identified the analog contents of your circuit design. The higher the value of `analog`, the more conservative the partition algorithm.

Note: This does not apply to `a` or `s` mode.

Table 3-5 analog Options

Option	Description
`analog=0`	Digital and memory circuits
`analog=1`	Digital, memory, and mixed signal circuits (default)
`analog=2`	Mixed signal, analog, and RF circuits
`analog=3`	Analog and RF circuits
`analog=4`	Mixed signal circuits (high sensitivity)

Applying `analog=2` or `analog=3` can slow down the simulation by forcing more conservative partitioning. To avoid slowing down the simulation, while maintaining accuracy on highly sensitive analog blocks, the analog option can be specified locally. Setting the option locally on sensitive analog blocks allows the simulator to keep the default analog level on the rest of circuit.

Example

Spectre Syntax:

```
usim_opt analog=2 inst=x1.xpll
```

SPICE Syntax:

```
.usim_opt analog=2 inst=x1.xpll
```

tells the Virtuoso UltraSim simulator to use a high-accuracy approach to analog simulation for feedback coupling in analog circuits.

Analog Autodetection

Virtuoso UltraSim simulator analog autodetection can be used to autodetect analog circuits (simulator automatically uses appropriate simulation settings for these circuits).

Note: Analog autodetection is limited to analog-to-digital conversion (ADC) and PLL circuit designs.

Description

Controls autodetection and promotion of analog circuits.

Table 3-6 search Options

Option	Description
`search=default`	Disables analog autodetection (default)
`search=analog`	Enables analog autodetection for ADC and PLL designs

Example

Spectre Syntax:

```
usim_opt search=analog
```

SPICE Syntax:

```
.usim_opt search=analog
```

tells the Virtuoso UltraSim simulator to enable autodetection of analog circuits.

Simulation Control Options

Operating Point Calculation Method

By default, the Virtuoso UltraSim simulator uses a pseudo-transient method of calculating the operating point. This method has been proven to handle the majority of circuits. It consists of two steps: First the power supplies are ramped and then the voltage levels are stabilized with a transient simulation. The simulator also allows you to skip the operating point calculation and to load an operating point from another simulation. In case the pseudo-transient method leads to problems, there is a pseudo-transient method available which only ramps up power supplies. The `dc` option is used to specify the operating point calculation method.

dc

Description

Defines the DC simulation algorithm the Virtuoso UltraSim simulator applies to the circuit.

Table 3-7 dc Options

Option	Description
`dc=0`	No operating point calculation. Similar to use initial conditions (UIC) in HSPICE (registered trademark of Synopsys, Inc.). Strictly enforced initial condition. For nodes without initial condition specified, the initial voltages are set to 0.
`dc=1`	Complete dynamic operating point calculation using pseudo-transient algorithm. Strictly enforces the initial conditions (default in `ms`, `da`, and `df` modes).
`dc=2`	Fast pseudo-transient circuit state ramp-up.
`dc=3`	Complete static operating point calculation using source-stepping algorithm (default in `a` and `s` modes and automatic switching to `dc=1` in case of non convergence). Initial conditions are forced on to nodes by using a voltage source in series with a resistor whose resistance is 1 ohm; default Virtuoso Spectre® *rforce* value. **Note:** `dc=3` is not recommended for `ms`, `df`, and `da` mode simulations.
`dc=4`	Complete pseudo-transient operating point (OP) calculation with damping. Suitable for designs including oscillators or designs where `dc=1` causes the DC calculation to exit prematurely.

In a transient analysis, the first calculation is a DC operating point using the DC equivalent model of a circuit. The DC operating point is then used as an initial estimate to solve the next time point in the transient analysis.

If dc=0, the Virtuoso UltraSim simulator sets the nodal voltages as defined by `.IC` statements (or by the `IC=` parameters in various element statements) instead of solving the quiescent operating point. The DC operating points of unspecified nodes are set to 0 volts.

Note: dc=0 is a Virtuoso UltraSim simulator feature and `usim_opt` option, and works for all netlist file formats supported by the simulator (see “Netlist File Formats” on page 51 for supported formats). For simulations based on a SPICE netlist file, setting dc=0 is equivalent to specifying `uic` in a `.tran` statement.

Example

Spectre Syntax:

```
usim_opt dc=0
```

SPICE Syntax:

```
.usim_opt dc=0
```

tells the simulator to skip the operating point calculation and to ramp up the power supplies during transient simulation.

Operating Point Calculation Control Options

By default, the Virtuoso UltraSim simulator uses a pseudo-transient method to calculate the DC operating point. The simulator exits the DC calculation when one of the following conditions occur: A stable operating point is reached, the number of DC events reach a certain limit, or the calculation time reaches the three hour limit. This method works for most circuits. For larger circuits, you can extend the DC calculation time by using the `dc_prolong` option.

If the DC calculation does not reach any of the aforementioned conditions, the Virtuoso UltraSim simulator issues a warning message and continues the simulation. You can also use the `dc_exit` option to stop the simulation if a stable solution is not reached (useful when the DC calculation is important for simulation accuracy).

dc_prolong

Description

Controls the exit criteria for operating point calculations.

Table 3-8 dc_prolong Options

Option	Description
`dc_prolong=0`	The Virtuoso UltraSim simulator exits the DC calculation when the three hour time limit is reached or when the number of DC events reaches a certain limit (default)
`dc_prolong=1`	The simulator extends the DC calculation until a stable operating point is reached

Example

Spectre Syntax:

```
usim_opt dc_prolong=1
```

SPICE Syntax:

```
.usim_opt dc_prolong=1
```

tells the Virtuoso UltraSim simulator to continue the DC calculation until a stable operating point is reached.

DC Options

Progress Report

The Virtuoso UltraSim simulator uses different DC methods to calculate the operating point. In general, the DC calculation is fast and does not require a progress report. When a large design is being simulated, and the DC calculation takes longer than five minutes (CPU time), the simulator prints a progress report. The report is printed for every progress_p percentage of the DC calculation time.

For dc=0, 2, and 3, DC calculation progress is reported in a single stage. For dc=1, the DC progress report is comprised of two stages. In cases where a stable solution is not reached in the second stage, the DC calculation continues and a DC steady factor is reported. This factor should get smaller, so it fits within 0 and 1 when the DC calculation approaches convergence.

dc_rpt_num

Description

The Virtuoso UltraSim simulator uses a pseudo-transient method to calculate the DC operating point and generally is able to provide a stable solution. In some cases, the DC calculation does not reach a stable state. For this situation, you can use the `dc_rpt_num` option to print unstable nodes to a `.dcr` file.

Note: The unstable nodes are only reported when `dc=1` or `dc=2` is specified, and the DC solution is not stable when the simulator completes the DC calculation.

Table 3-9 dc_rpt_num Options

Option	Description
`dc_rpt_num=0`	Unsettled nodes are not reported (default)
`dc_rpt_num=`*`value`*	Reports values for the most unstable nodes in order of DC steady state factor (integer, unitless) **Note:** The DC steady state factor describes how close the calculated DC value for a node is to a stable DC solution.

Example

Spectre Syntax:

```
usim_opt dc_rpt_num=20
```

SPICE Syntax:

```
.usim_opt dc_rpt_num=20
```

tells the Virtuoso UltraSim simulator to print 20 unstable nodes in order of DC steady state factor.

dc_exit

Description

Controls the exit criteria for DC calculations if a stable solution is not reached.

Table 3-10 dc_exit Options

Option	Description
`dc_exit=0`	The Virtuoso UltraSim simulator continues the simulation after issuing a warning message if the DC calculation does not reach a stable solution (default)
`dc_exit=1`	The simulator stops the simulation after issuing an error message if the DC calculation does not reach a stable solution

Note: Setting dc=0 and dc=2 does not provide a stable DC solution and produces an error condition in the Virtuoso UltraSim simulator when `dc_exit` is set to 1 (instead use `dc_exit=0` to run `dc=0/2`, or set `dc=1/3`).

Example

Spectre Syntax:

```
usim_opt dc_exit=1
```

SPICE Syntax:

```
.usim_opt dc_exit=1
```

tells the simulator to exit the simulation when the DC calculation does not reach a stable operating point.

Integration Method

The Virtuoso UltraSim simulator offers different choices for the ordinary differential equation (ODE) solver to integrate the circuit equation. The order of the integration method determines the rate of decay of numerical error. The first-order Backward Euler method is a good choice for simulations with sharp waveforms, while the second-order Trapezoidal method and Gear method are good choices for simulations with smooth waveforms. Both methods switch automatically to Backward Euler if convergency problems occur. The Trapezoidal method has no artificial numerical damping and might be a good choice for simulating oscillators if oscillators cannot start oscillation with other methods. This is even more true for the strictly Trapezoidal method, which does not switch to Euler. In general, second-order methods are faster than first-order method when a relatively tight tolerance is desired and the waveforms have big regions that are smooth.

method

Description

Defines the integration method the Virtuoso UltraSim simulator applies to the circuit.

Table 3-11 method Options

Option	Description
`method=euler`	First-order backward Euler method (default if `sim_mode=ms`/ `da`/`df`/`dx`)

Table 3-11 method Options, *continued*

Option	Description
`method=trap`	Trapezoidal method (automatic switching)
`method=gear2`	Second-order gear method (automatic switching–default if `sim_mode=s` or `a`)
`method=traponly`	Strictly trapezoidal method (no automatic switching)
`method=gear2only`	Strictly second-order gear method (no automatic switching)

Example

Spectre Syntax:

```
usim_opt method=gear2
```

SPICE Syntax:

```
.usim_opt method=gear2
```

tells the simulator to use the second-order `gear` integration method to integrate the circuit equations.

Notes

- When convergency problems occur, the Virtuoso UltraSim simulator automatically switches back to the Euler method.
- Only `dx` mode supports method=euler.

Simulation Tolerances

Although the `speed` option is all that is commonly needed to control the general accuracy of the Virtuoso UltraSim simulator, individual simulation options can be set for more fine grained control over the speed versus accuracy trade-off. You can set parameters for the universal relative tolerance `tol`, the absolute voltage tolerance `abstolv`, the absolute current tolerance `abstoli`, the local truncation error (LTE) `trtol`, and the maximum step size `maxstep_window`. Table 3-9 shows how these parameters depend on the `speed` settings.

Table 3-12 Simulation Tolerance Parameters

speed	1	2	3	4	5	6	7	8
`abstoli`	1pA	1pA	1pA	1pA	1pA	1pA	1pA	1pA

Table 3-12 Simulation Tolerance Parameters, *continued*

speed	1	2	3	4	5	6	7	8
`abstolv`	1µV	1µV	1µV	1µV	1µV	1µV	1µV	1µV
`tol`	0.0001	0.001	0.0025	0.005	0.01	0.02	0.04	0.07
`trtol`	7	7	5.6	4.6	3.5	3.5	3.5	3.5

abstoli

Description

`abstoli` is the absolute tolerance for currents and defines the smallest current of interest in the circuit. Currents smaller than `abstoli` are ignored in convergence checking and time step control.

Table 3-13 abstoli Option

Option	Description
`abstoli=`*`value`*	Absolute tolerance (double, unit A, 0 < value < 1, default 1 pA)

Example

Spectre Syntax:

```
usim_opt abstoli=1e-11
```

SPICE Syntax:

```
.usim_opt abstoli=1e-11
```

tells the simulator to use an absolute current tolerance of 10 pA for current calculations.

abstolv

Description

`abstolv` is the absolute tolerance for voltages and defines the smallest voltage of interest in the circuit. Voltages smaller than `abstolv` are ignored in convergence checking and time step control. Generally, the absolute voltage tolerance is set 10^6 to 10^8 times smaller than the largest voltage signal.

Table 3-14 abstolv Option

Option	Description
`abstolv=`*`value`*	Absolute tolerance (double, unit V, 0 < value < 1, default 1 uV)

Example

Spectre Syntax:

```
usim_opt abstolv=1e-7
```

SPICE Syntax:

```
.usim_opt abstolv=1e-7
```

tells the simulator to use a absolute voltage tolerance of 0.1 uV for voltage calculations.

maxstep_window

Spectre Syntax

```
usim_opt maxstep_window=[ time1 maxstep1 time2 maxstep2 time3 maxstep3... ]
```

SPICE Syntax

```
.usim_opt maxstep_window=[ time1 maxstep1 time2 maxstep2 time3 maxstep3... ]
```

Description

`maxstep_window` is used to specify the maximum time step over different simulation time windows. The simulation time window is specified in the square brackets [ ] as pairs of numbers. For each pair, the first number is the start time for the simulation time window and the second number is the maximum time step for this window ending with the next time point. That is, the `maxstep_window` value for the simulation time window from `time1` to `time2` is `maxstep1`, `time2` to `time3` is `maxstep2`, and so forth.

Note: The time points can only use sequential double values (for example, `time1` < `time2` < `time3`).

Table 3-15 maxstep_window Options

Option	Description
maxstep1 <maxstep2...>	Maximum time step (in seconds; no default)
time1 <time2...>	Simulation window time point (in seconds; no default)

Examples

In the following Spectre syntax example

```
usim_opt maxstep_window=[ 0 1n 1u 1p 10u 1e20 ]
```

tells the Virtuoso UltraSim simulator the maximum time step is `1n` seconds during simulation time window 0 to 1u, 1p seconds during simulation time window 1 to 10u, and after simulation time 10u, the maximum time step is set to `1e20` seconds (large number indicating no maximum time step control).

In the following SPICE syntax example

```
.usim_opt maxstep_window=[ 100u 1p 200u 1e20 ] x1.x2
```

sets the maximum time step to `1p` during simulation time window 100 u~200 u and 1e20 after 200 u. This setting applies only to instance `x1.x2`.

tol

Description

The relative tolerance `tol` is used as the universal accuracy control in the Virtuoso UltraSim simulator. Except for extremely small signals, the relative tolerance is the dominating criterion in the transient simulation. A value between 0 and 1 can be chosen; values closer to zero imply greater accuracy. `tol` determines the upper limit on errors relative to the size of the signal. In case you need to use a relative tolerance, which cannot be set by the high-level `speed` option, the `tol` option can be used to adjust the relative tolerance.

Table 3-16 tol Option

Option	Description
`tol=`*value*	Double, 0 < value < 1 (default 0.01)

Example

Spectre Syntax:

```
usim_opt tol=0.005
```

SPICE Syntax:

```
.usim_opt tol=0.005
```

tells the simulator to use a relative tolerance of 0.005 for current and voltage calculation.

trtol

Description

`trtol` is used in the LTE criterion, where it multiplies `reltol`. It it set to `3.5` by default, and should not be changed for most circuits.

Table 3-17 trtol Option

Option	Description
`trtol=`*`value`*	Error criterion (double, 1 < value < 14, default 3.5)

Example

Spectre Syntax:

```
usim_opt trtol=8
```

SPICE Syntax:

```
.usim_opt trtol=8
```

tells the simulator to use trtol=8.

Simulation Convergence Options

For circuits that have difficulty converging during simulation, as a result of the design or model being used, you can use the `gmin_allnodes` or `cmin_allnodes` options to assist in convergence. The effectiveness of a particular option is dependent on the type of circuit used in the simulation. Cadence recommends trying one or both options to solve the convergence problem.

gmin_allnodes

Description

Adds the specified conductance to each node.

Table 3-18 gmin_allnodes Option

Option	Description
`gmin_allnodes=`*`value`*	Adds the specified conductance to each node (default is zero)

Example

Spectre Syntax:

```
usim_opt gmin_allnodes=1e-10
```

SPICE Syntax:

```
.usim_opt gmin_allnodes=1e-10
```

tells the Virtuoso UltraSim simulator to add a conductance of 1e-10 mho to each node.

cmin_allnodes

Description

Adds the specified capacitance to each node.

Table 3-19 cmin_allnodes Option

Option	Description
`cmin_allnodes=`*`value`*	Adds the specified capacitance to each node (default is zero)

Example

Spectre Syntax:

```
usim_opt cmin_allnodes=1e-15
```

SPICE Syntax:

```
.usim_opt cmin_allnodes=1e-15
```

tells the simulator to add a capacitance of 1 fF to each node.

Save and Restart

Spectre Syntax

```
usim_save <file=”dir/filename”> <time=[time1,time2]> > <repeat=save_period>
```

SPICE Syntax

```
.usim_save <file=”dir/filename”> <time=[time1,time2]> > <repeat=save_period>
```

Description

The Virtuoso UltraSim simulator save (`usim_save`) and restart (`usim_restart`) features allow you to save the simulation database at a specified time point. The simulation database can be used to restart the simulation at that time point. Applications of save and restart include:

- Achieve maximum simulation speed by only simulating the portion of time that requires a highly accurate simulation mode (for example, simulate a PLL locking process in accurate mode and then switch to a higher speed mode once the PLL is locked)
- Perform “what if” analyses of problematic sections of a design
- Test circuits that are only semi-functional by using an abstract model for capabilities not implemented
- Support rapid simulation of circuits by using behavioral models during non-critical accuracy phases of simulation
- Use full-chip simulations as test bench generators for block simulations
- Experiment with different simulation options on sections of the circuit or on the entire circuit (for example, `sim_mode`, `speed`, or output flushing)
- Replace portions of the circuit, set the simulator to use the existing port voltages as integrated circuits (ICs) for the replaced circuit, initialize the new circuit, and run the simulation

Use Model

- You can invoke the save and restart options using netlist file commands or during an interactive run.
- In subsequent simulations, changes to the circuit topology can add or delete nodes. The added nodes are initialized as if the operating points were not saved, and references to deleted nodes are ignored. The coincidental nodes are initialized to values saved from the previous simulation run.
- If a parameter or temperature sweep is performed, only the first operating point is saved.

  For example, if the input netlist file contains the statement

  ```
  .temp -10 0 25
  ```

  the operating point that corresponds to .temp -10 is saved.

Table 3-20 .usim_save Commands

Command	Description
`dir/filename`	Name of the file used to save the simulation state. Multiple `time` points are assigned unique names. For example, `filename@time1`, `filename@time2`, and `filename@time3`. The saved files contain the Virtuoso UltraSim version number. (Default is `<design>.save@time`).
`time1, time2`	Time at which the operating point is saved. A valid transient analysis statement is required to successfully save an operating point. (Default: 0).
`repeat`	Saves the operating point at specific intervals. For example, t=save_time1+N*save_period, N=0,1,2,... If `repeat` is used, subsequent `save_time` inputs are discarded. The saved files are named `save_file@t`. (Default: Save only once).

Spectre Syntax

```
usim_restart file="dir/load_file"
```

SPICE Syntax

```
.usim_restart file="dir/load_file"
```

Table 3-21 .usim_restart Commands

Command	Description
`dir/load_file`	Name of the file that contains the saved simulation state. (Default: `<design>.simsave`).

Strobing Control Options

Description

The strobing function is used to select the time interval between the data points that the Virtuoso UltraSim simulator saves. It is enabled by setting the `strobe_period` option. The simulator forces a time step for each point it saves, so data is computed instead of interpolated, improving the accuracy of post simulation FFT analysis.

The strobe options are documented in the following table.

Table 3-22 strobe Options

Option	Description
`strobe_period`	Sets the time interval between data points saved by the simulator
`strobe_start`	Sets the strobing start time (optional)
`strobe_stop`	Sets the strobing stop time (optional)
`strobe_delay`	Sets the delay time between `strobe_start` and `strobe_stop` (optional – default is `0`)

Example

Spectre Syntax:

```
usim_opt strobe_period=10n strobe_start=1u strobe_delay=5n
```

SPICE Syntax:

```
.usim_opt strobe_period=10n strobe_start=1u strobe_delay=5n
```

tells the simulator to start strobing at time=1 us, to save data points at 10 ns intervals, and to continue strobing until the end of the transient simulation. The first actual strobing occurs at time=1.005 us.

Automatic Strobing for Spectre Fourier Elements

When using a Spectre Fourier element in a circuit design, the Virtuoso UltraSim simulator automatically activates the strobing function to improve Fourier analysis accuracy. The strobe period is set equal to the period of the fundamental frequency divided by 1024 or to the number of points in the Fourier analysis (simulator uses the larger number of the two methods).

Modeling Options

To address all types of simulation, ranging from high-speed digital simulation to high-precision analog simulation, the Virtuoso UltraSim simulator offers a variety of MOSFET models covering different levels of abstraction. Although the `sim_mode` option is what is commonly needed for controlling device modeling in the simulator, individual model options can be set for more fine grained control over the trade-off between speed and accuracy.

MOSFET Modeling

The Virtuoso UltraSim simulator options `mos_method` and `mosd_method` are used to control MOSFET modeling. While the BSIM SPICE model uses one set of equations for the MOSFET device, the representative models for `dx`, `df`, `da`, `ms`, and `a` mode use different models for the core device (current and charge model), and the diffusion junctions of the MOSFET. The `mos_method` option determines the core device model, and the `mosd_method` option defines the diffusion model. If `mos_method` is set to SPICE, the option `mosd_method` is ignored. Table 3-23 on page 184 gives an overview of the type of model used by each simulation mode or each `mos(d)_method` option.

Table 3-23 Simulation Model Modes

	Current Model (mos_method)	Charge Model (mos_method)	Diffusion (mosd_method)
`df/dx`	Nonlinear digital model	Constant capacitance	Constant capacitance
`da`	Nonlinear digital model	Nonlinear model	Constant capacitance (same as `df`)
`ms|a`	Analog model	Analog model	Analog model

Table 3-23 Simulation Model Modes, *continued*

	Current Model (mos_method)	Charge Model (mos_method)	Diffusion (mosd_method)
`s`	BSIM SPICE	BSIM SPICE	-

mos_method

Description

Defines the MOSFET current and charge modeling.

Table 3-24 mos_method Options

Option	Description
`mos_method=df`	Nonlinear digital representative current and constant capacitance charge models used in `df` and `dx` modes
`mos_method=da`	Nonlinear digital representative current and charge model
`mos_method=a`	Nonlinear analog current and charge model
`mos_method=s`	BSIM SPICE MOSFET model

Example

Spectre Syntax:

```
usim_opt mos_method=a
```

SPICE Syntax:

```
.usim_opt mos_method=a
```

tells the simulator to use the nonlinear analog current and charge model for all MOSFET devices.

mosd_method

Description

Defines the MOSFET diffusion junction modeling. If `mos_method` is set to `s`, `mosd_method` is ignored.

Table 3-25 mosd_method Options

Option	Description
`mosd_method=df`	Constant capacitance model for diffusion junction
`mosd_method=a`	Nonlinear analog model for diffusion junction

Example

Spectre Syntax:

```
usim_opt mosd_method=a
```

SPICE Syntax:

```
.usim_opt mosd_method=a
```

tells the simulator to use the nonlinear analog model for all MOSFET diffusion junctions.

Note: The `mos_method` and `mosd_method` options cannot be changed for design blocks simulated in `dx` mode.

mos_cap

Description

Defines the MOSFET core device capacitance model in `a` or `ms` mode. A linear model can provide significant performance improvements over a nonlinear model. Cadence recommends using `mos_cap` only for designs that are not sensitive to nonlinear device capacitances.

Table 3-26 mos_cap Options

Option	Description
`mos_cap=nl`	The Virtuoso UltraSim simulator uses nonlinear MOSFET device capacitances (default)
`mos_cap=lin`	The simulator uses linear MOSFET device capacitances

mod_a_isub

Description

Defines the modeling of substrate current for BSIM3v3, BSIM4, BSIMSOI, and SSIMSOI devices. If `s` mode is used, the Virtuoso UltraSim simulator considers substrate current automatically.

Note: This option is only applicable to analog representative models (not applicable to `da` or `df` mode).

Table 3-27 mod_a_isub Options

Option	Description
`mod_a_isub=0`	No substrate current in the analog representative model (substrate current is ignored)
`mod_a_isub=1`	The simulator determines the need for modeling substrate current in the analog representative model, based on current value (default)
`mod_a_isub=2`	Substrate current is included in the analog representative model, even if the current is small

Example

Spectre Syntax:

```
usim_opt mod_a_isub=1
```

SPICE Syntax:

```
.usim_opt mod_a_isub=1
```

tells the simulator to activate `isub` during the simulation if it determines `isub` is large enough to be considered.

mod_a_igate

Description

Defines the modeling of gate current for BSIM4, BSIMSOI, and SSIMSOI devices. If `s` mode is used, the Virtuoso UltraSim simulator considers gate current automatically.

Note: This option is only applicable to the analog representative model (not applicable to `da` or `df` mode).

Table 3-28 mod_a_igate Options

Option	Description
`mod_a_igate=0`	Gate leakage current is ignored in the analog representative model (default)
`mod_a_igate=1`	The simulator determines the need for modeling gate current in the analog representative model, based on current value
`mod_a_igate=2`	Gate current is modeled, even if the current is small

Example

Spectre Syntax:

```
usim_opt mod_a_igate=2
```

SPICE Syntax:

```
.usim_opt mod_a_igate=2
```

tells the simulator to activate gate current in the analog representative model for a current of any size.

table_mem_control

Description

Controls the memory usage of table models. Enable this option to avoid excessive use of memory due to table models.

Table 3-29 table_mem_control Options

Option	Description
`table_mem_control=0`	No control on memory usage of table models (default).
`table_mem_control=1`	The Virtuoso UltraSim simulator controls the memory usage for table models.

Examples

Spectre Syntax:

```
usim_opt table_mem_control=1
```

SPICE Syntax:

```
.usim_opt table_mem_control=1
```

tells the Virtuoso UltraSim simulator to control the memory usage of table models.

Analog Representative Model for Generic MOSFET Devices

Spectre Syntax

```
usim_opt generic_mosfet=device_master_name
```

SPICE Syntax

```
.usim_opt generic_mosfet=device_master_name
```

Note: `device_master_name` is a string.

Description

The Virtuoso UltraSim simulator supports building an analog representative model for generic MOSFET devices, allowing you to treat a generic MOSFET device as a “black box.” It is useful for building an analog representative model for proprietary MOSFET devices. This requires the generic MOSFET to be implemented via the compiled-model interface (CMI).

Limitations

- The MOSFET device cannot have more than four external terminals, and the terminals must be placed in the following order: D, G, S, and B.
- The MOSFET device can have two internal nodes, arranged in the following order: Internal S and internal D (default). If the order of the internal nodes is reversed, use `usim_opt mosfet_sd=0` (default is `1`).

Example

Spectre Syntax:

```
usim_opt generic_mosfet=VMOS
usim_opt sim_mode=ms
```

SPICE Syntax:

```
.usim_opt generic_mosfet=VMOS
.usim_opt sim_mode=ms
```

The Virtuoso UltraSim simulator builds an analog representative model for the MOSFET devices with the master name `VMOS`.

Diode Modeling

diode_method

Description

Defines diode modeling in the Virtuoso UltraSim simulator, with an emphasis on juncap modeling.

Table 3-30 diode_method Option

Option	Description
`diode_method=df`	Constant capacitance model for juncap only

Table 3-30 diode_method Option

Option	Description
`diode_method=a`	Nonlinear analog model for juncap only (default for juncap by `a`/`ms`/`da`/`df` modes)
`diode_method=s`	Berkeley SPICE diode model (default for diode in all simulation modes, except for juncap)

Note: For juncap, the default is `s` for `s` mode and `a` for `a`/`ms`/`da`/`df` modes.

Example

Spectre Syntax:

```
usim_opt diode_method=a model=d
```

SPICE Syntax:

```
.usim_opt diode_method=a model=d
```

tells the simulator to use the default model if `d` is a juncap model (if `d` is a regular diode, the `diode_method` option is ignored by the simulator).

dcut

Description

The `dcut` option deletes all or selected diodes in the netlist file. This is helpful in designs with large amounts of diodes, where the diodes do not have an impact on the function of the design (for example, input protection diodes).

Table 3-31 dcut Option

Option	Description
`dcut=0`	No diodes deleted (default)
`dcut=1`	Diodes deleted

Example

Spectre Syntax:

```
usim_opt dcut=1 inst=x1.x2
```

SPICE Syntax:

```
.usim_opt dcut=1 inst=x1.x2
```

tells the simulator to delete all the diodes in `x1.x2` and all its subcircuits.

minr

```
optionname options minr=value
```

Description

This option allows you to short small resistors in models (for example, diode, bjt, and BSIM3 models).

Note: `minr` is only valid in Spectre format.

Example

```
simulator lang=spectre
option1 options minr=1.0e-4
```

tells the Virtuoso UltraSim simulator to short all resistors <1.0e-4 in all models.

Operating Voltage Range

The Virtuoso UltraSim simulator uses representative digital models in `df` and `da` mode. These models are generated in the beginning of the simulation, stored in the `*.lsn` file, and can be reused using the `model_lib` option. The `.lsn` file gets updated when changes in the devices, voltage supplies, or process variations occur. The voltage range used for building the models is automatically chosen by detecting the value of the highest power supply and is used for all device models. The generated models are valid over at least 2 times the given voltage range.

In designs with low and high voltage devices, where the voltages differ by an order of magnitude (that is, 2 V/10 V), using higher voltage to build the low voltage device models can lead to a significant modelling error. In this case, it is recommended to use the lower voltage for the low voltage devices and the higher voltage for the high voltage devices. To specify the voltage range, the Virtuoso UltraSim simulator provides the `vdd` option, which can be applied to devices, subcircuits, and instances.

vdd

Description

Defines the maximum voltage for the generation of digital representative models (`da` and `df` mode only).

Table 3-32 vdd Option

Option	Description
`vdd=`*`value`*	Maximum voltage (double, unit V, default max. vdd)

Example

Spectre Syntax:

```
usim_opt vdd=2.1 model=[nf pf]
```

SPICE Syntax:

```
.usim_opt vdd=2.1 model=[nf pf]
```

tells the simulator to use 2.1 volts as the maximum voltage for the model generation of device models `nf` and `pf`.

Treatment of Analog Capacitors

The Virtuoso UltraSim simulator uses partitioning to speed up the simulation in `df`, `da`, and `ms` modes. Partitions are built by putting all channel-connected devices into the same partition, and by cutting between capacitive coupled nodes. This approach works fine for digital circuits, memories, and most mixed signal applications.

In some analog circuits, the coupling is designed as a functional part of the circuit (that is, charge pumps), or it strongly affects the functionality of the circuit. In this case, the two circuits connected by the analog coupling capacitance need to be simulated in the same partition.

The `canalog` and `canalogr` options determine the thresholds for identifying analog coupling capacitances. Any capacitor larger than `canalog`, and its ratio to the total capacitance at either node is greater than `canalogr`, is treated as an analog capacitance. This same threshold applies to nonlinear capacitances (for example, MOSFET Cgd). Setting `canalog` and `canalogr` lower can lead to more stable and accurate results, but usually increases run time. Setting it too high can cause less accurate results when heavy coupling occurs.

canalog

Description

Defines the absolute threshold value for identifying analog coupling capacitances in `df`, `da`, and `ms` modes (does not apply to `a` or `s` mode).

Table 3-33 canalog Option

Option	Description
`canalog=`*`value`*	Maximum capacitance value (double and unit F) The `canalog` default value is dependent on the value of the analog option: ■ If `analog=0`, then `canalog=100f` ■ If `analog=1`, then `canalog=100f` ■ If `analog=2`, then `canalog=30f` ■ If `analog=3`, then `canalog=10f`

canalogr

Description

Defines the relative threshold value for identifying analog coupling capacitances in `df`, `da`, and `ms` modes (does not apply to `a` or `s` mode).

Table 3-34 canalogr Option

Option	Description
`canalogr=`*`value`*	Relative threshold value (double, unit F, and 0 < value < 1) The `canalogr` default value is dependent on the value of the analog option: ■ If `analog=0`, then `canalogr=0.49` ■ If `analog=1`, then `canalogr=0.45` ■ If `analog=2`, then `canalogr=0.35` ■ If `analog=3`, then `canalogr=0.25`

Example

Spectre Syntax:

```
usim_opt canalogr=0.1
```

SPICE Syntax:

```
.usim_opt canalogr=0.1
```

tells the simulator to treat every capacitor larger than 0.1 pF, and canalogr=0.1 bigger than 10% of the nodes capacitance on either side, as analog capacitance.

Inductor Shorting

The Virtuoso UltraSim simulator supports the simulation of inductances. Simulations including inductors can be more time consuming. Sometimes it is helpful to short all inductors in a netlist file, to do a first functional verification. The options `lshort` and `lvshort` provide the opportunity to short inductors in the signal paths or power supply lines.

lshort

Description

Defines the threshold value for inductor shorting in signal nets. Inductors smaller than *value* are shorted.

Table 3-35 lshort Option

Option	Description
`lshort=`*`value`*	Inductor value (double, unit H, 0 < value, default 0)

Example

Spectre Syntax:

```
usim_opt lshort=1µ
```

SPICE Syntax:

```
.usim_opt lshort=1µ
```

tells the simulator to short all inductors less than 1µH in signal nets.

lvshort

Description

Defines the threshold value for inductor shorting in power nets. Inductors smaller than *value* are shorted.

Table 3-36 lvshort Option

Option	Description
`lvshort=`*`value`*	inductor value (double, unit H, 0 < value, default 0)

Example

Spectre Syntax:

```
usim_opt lvshort=1µ
```

SPICE Syntax:

```
.usim_opt lvshort=1μ
```

tells the simulator to short all inductors less than 1μH in power nets.

Waveform File Format and Resolution Options

Waveform Format

wf_format

The Virtuoso UltraSim simulator supports SignalScan Turbo 2 (SST2), fast signal database (FSDB), parameter storage format (PSF), and waveform data format (WDF). It can generate SST2 format for viewing in SimVision and Virtuoso Visualization and Analysis (ViVA), FSDB for nWave, PSF format for ViVA, and WDF for viewing with the Sandwork WaveView Analyzer.

Note: The recommended SimVision waveform viewer can be downloaded from the latest Cadence IUS release (the SimVision license is included in the Virtuoso UltraSim simulator license file).

Table 3-37 wf_format Options

Option	Description
`wf_format=`*`sst2`*	SST2 format (SimVision and ViVA waveform viewers; `trn/dsn`; default)
`wf_format=`*`fsdb`*	FSDB format (nWave waveform viewer; `fsdb`)
`wf_format=`*`psf`*	PSF format (ViVA waveform viewer; `tran`)
`wf_format=`*`wdf`*	WDF format (Sandwork WaveView Analyzer; `wdf`)
`wf_format=`*`psfxl`*	PSF XL format (ViVA waveform viewer)

The Virtuoso UltraSim simulator is able to write files of unlimited size in SST2 and FSDB format, whereas PSF and WDF formats are limited to a maximum of 2 GByte files. Use the `wf_maxsize` option to split waveform files.

Data compression varies between the formats: SST2 – high, FSDB and WDF – medium, and PSF – low. It is recommended that you use SST2 format for larger circuit designs.

PSF XL is a new Cadence waveform format supported in ViVA (available in IC 6.1.3 release) which provides a high compression rate for large circuit designs. RTSF is a new PSF extension and provides improved viewing performance in ViVA (available in IC 6.1.2 and later releases). RTSF only applies to PSF and PSF XL, and it can be enabled by using `+rtsf` on the command line.

The Virtuoso UltraSim simulator writes waveform files into the current directory. To enable other Cadence tools to read Virtuoso UltraSim PSF format, create a raw directory using the Virtuoso UltraSim simulator `-raw` command line option

```
ultrasim pll.scs -raw pll.raw
```

In the following Spectre syntax example,

```
usim_opt wf_format=psf
```

tells the simulator to generate a waveform file in PSF format.

In the following SPICE syntax example,

```
.usim_opt wf_format=[psf sst2]
```

tells the simulator to generate two waveform files, one in PSF format and the other in SST2 format.

Updating Waveform Files

The Virtuoso UltraSim simulator allows you to specify after what period of transient simulation time the waveform data is printed into the output waveform file, determined by the option `dump_step`. Its default value is 10% of trend. You can also enter the interactive mode with `Control-C`, and use the interactive command flush any time.

dump_step

Description

Defines the time period after which the waveform data is printed into the output waveform file.

Table 3-38 dump_step Option

Option	Description
`dump_step=`*`value`*	Time period (double, unit s, 0 < value, default 10% of tend)

Example

Spectre Syntax:

```
usim_opt dump_step=10n
```

SPICE Syntax:

```
.usim_opt dump_step=10n
```

tells the simulator to print waveforms every 10 ns of transient time into the output waveform file.

Waveform File Size

wf_maxsize

Description

There are specific waveform formats (for example, *psfbin* or *psfascii*) with 2 Gigabyte file size limitations. The `wf_maxsize` option is used to limit the maximum size of a waveform output file. If this option is not set, and the output file exceeds its size limit, the simulation stops.

Table 3-39 wf_maxsize Option

Option	Description
`wf_maxsize=`*[number]*	Defines the maximum size of the output file. If the maximum size is exceeded, the Virtuoso UltraSim simulator splits the file into multiple, smaller files.

Example

Spectre Syntax:

```
usim_opt  wf_maxsize=1e9
```

SPICE Syntax:

```
.usim_opt  wf_maxsize=1e9
```

tells the simulator to limit the output file size to 1 Gigabyte (if the file size is exceeded, it is split into two files identified by generic names).

Note: If a circuit.sp file and PSF waveform format is used, the following output file list is generated: circuit.tran, circuit_1.tran, circuit_2.tran, circuit_3.tran ... *circuit_n.tran*.

Waveform File Resolution

The accuracy for voltage and current waveforms, and the time resolution in the output waveform file can be set individually, depending on the application. The Virtuoso UltraSim simulator provides the absolute criteria wf_abstoli, wf_abstolv, and wf_tres, and the relative tolerance wf_reltol.

When plotting a waveform, the next point is determined by the relative change compared to the individual signal, or by the absolute change in the waveform. Except for extremely small signals, the relative criterion is dominant. The time resolution also determines the time unit of the waveform file.

The implemented solution is usually sufficient for any application. You need to verify that the resolution is appropriate for your design. This is especially important because all measurement functions are based on the resulting waveforms.

wf_filter

Description

Enables customized filtering of waveform data. In default ms mode, the Virtuoso UltraSim simulator uses moderate filtering (wf_filter=2) to minimize waveform file size without losing accuracy for standard applications. For sensitive analog designs and small signal amplitudes, no filtering (wf_filter=0) or conservative filtering (wf_filter=1) may be required. Greater waveform file size reduction for large digital and memory designs can be achieved using wf_filter=3 and wf_filter=4.

Table 3-40 wf_filter Options

Option	Description
wf_filter=0	Waveform data filter disabled (default in s mode)
wf_filter=1	Conservative waveform data filter for analog circuits with small signal amplitudes (default in a mode)
wf_filter=2	Moderate waveform data filter (default in ms, da, and df modes)

Table 3-40 wf_filter Options, *continued*

Option	Description
`wf_filter=3`	Aggressive waveform filtering for timing verification of large memory designs, digital circuits, and some mixed signal designs
`wf_filter=4`	Aggressive waveform filtering for functional verification of large memory designs and digital circuits

Example

Spectre Syntax:

```
usim_opt wf_filter=0
```

SPICE Syntax:

```
.usim_opt wf_filter=0
```

tells the simulator not to filter the waveform data.

wf_reltol

Description

Defines the relative current and voltage criterion for the waveform plot in the output waveform file.

Table 3-41 wf_reltol Option

Option	Description
`wf_reltol=`*`value`*	Relative criterion (double, unitless, 0 < value < 1)

Example

Spectre Syntax:

```
usim_opt wf_reltol=0.001
```

SPICE Syntax:

```
.usim_opt wf_reltol=0.001
```

tells the simulator to print the next point of a waveform in the output waveform file, if the change in the waveform is 0.1% (and the absolute criterion does not apply).

wf_tres

Description

Defines the time resolution and time unit in the output waveform file.

Table 3-42 wf_tres Option

Option	Description
`wf_tres=`*`value`*	Time resolution (double, unit s, 0 < value, default 1ps)

Example

Spectre Syntax:

```
usim_opt wf_tres=10p
```

SPICE Syntax:

```
.usim_opt wf_tres=10p
```

tells the simulator to use a time resolution and unit of 10 ps in the output waveform file.

wf_abstolv

Description

Defines the absolute voltage resolution in the output waveform file.

Table 3-43 wf_abstolv Option

Option	Description
`wf_abstolv=`*`value`*	Voltage resolution (double, unit V, 0 < value)

Example

Spectre Syntax:

```
usim_opt wf_abstolv=0.01m
```

SPICE Syntax:

```
.usim_opt wf_abstolv=0.01m
```

tells the simulator to use a voltage resolution of 0.01 mV in the output waveform file.

wf_abstoli

Description

Defines the absolute current resolution in the output waveform file.

Table 3-44 wf_abstoli Option

Options	Description
`wf_abstoli=`*`value`*	Current resolution (double, unit A, 0 < value)

Example

Spectre Syntax:

```
usim_opt wf_abstoli=1p
```

SPICE Syntax:

```
.usim_opt wf_abstoli=1p
```

tells the simulator to use a current resolution of 1 pA in the output waveform file.

Table 3-45 gives an overview of the default values for `wf_reltol`, `wf_abstolv`, and `wf_abstoli` dependent on the `wf_filter` option used.

Table 3-45 Waveform Filtering Options (Default Values)

wf_filter	wf_reltol	wf_abstolv	wf_abstoli
0	Not applicable (N/A)	N/A	N/A
1	1e-7	1e-6	1e-12
2	min{tol, 0.005}	1e-6	1e-12
3	min{tol, 0.005}	1e-3	1e-9

Table 3-45 Waveform Filtering Options (Default Values), *continued*

wf_filter	wf_reltol	wf_abstolv	wf_abstoli
4	min{tol, 0.005}	1e-2	1e-6

Node Name Format Control

wf_output_format

Description

Controls the appearance of a hierchical node name in the waveform database.

Table 3-46 wf_output_format Options

Option	Description
`wf_output_format=spice`	Includes the following format in the waveform database: `x1.x11.v(n1), x1.x11.i1(m1)`
`wf_output_format=spectre`	Includes the following formats in the waveform database: `x1.x11.n1` and `x1.x11.m1:1`
`wf_output_format=verilog`	Includes the following formats in the waveform database: `x1.x11.n1` and `x1.x11.m1:1_$flow`
`wf_output_format=spice_raw`	Includes the following formats in the waveform database: `v(x1.x11.n1)` and `i1(x1.x11.m1)`

Miscellaneous Options

Model Library Specification

The option `model_lib` specifies the name of the model library file that stores all digital representative models (the model library is always given a `.lsn` extension). The default name for the model library is the netlist filename. For example, suppose the netlist filename is `netlist.sp`. Then the model library file would be called `netlist.lsn`. It is recommended to keep the path of the file relative to the working directory.

This option is useful only if a large number of representative models are going to be built. The time for building representative models is not dominant compared to simulation. Occasionally the model build time can dominate the run time. Normally, you do not need to use this option because the Virtuoso UltraSim simulator automatically builds a model library file which is automatically loaded the next time you run the simulator. The first simulator run on a particular netlist file is somewhat slower than subsequent runs in the same directory.

If you want to reuse the representative models for different netlist files, or the same netlist file in different locations, you need to specify the library file to write to and read from. By using the same model library name and storing it in a central location, you can reuse models from many netlist files.

model_lib

Description

Defines the library file used for digital representative models (`da` and `df` mode only).

Table 3-47 model_lib Option

Option	Description
`model_lib=`*`filename`*	Filename (default `netlist.lsn`)

Examples

In the following Spectre syntax example

```
usim_opt model_lib=mod.lsn
```

tells the simulator to use the model file mod.lsn out of the netlist file directory.

In the following SPICE syntax example

```
.usim_opt model_lib="/home/user/ms2/mod.lsn"
```

tells the simulator to use the model file */home/user/ms2/mod.lsn*.

Warning Settings

The Virtuoso UltraSim simulator allows you to customize how warning messages are handled by the simulator. The number of messages per warning category can be limited globally for all warnings (`usim_opt warning_limit`) or individually for each category (`usim_report warning_limit`). When the specified category limit is reached, the simulator notifies you that the warning messages are no longer being displayed. Dangling and floating node warnings are controlled by the number of reported nodes.

warning_limit

Description

Limits the number of warnings issued per warning category and is applied globally to all warning messages. This option needs to be defined at the beginning of the netlist file.

Table 3-48 warning_limit Option

Option	Description
`warning_limit=`*`value`*	Number of warnings (integer, unitless; default is 5)

A limit can also be applied to a specific warning category using the `usim_report warning_limit` command.

Example

Spectre Syntax:

```
usim_opt warning_limit=10
```

SPICE Syntax:

```
.usim_opt warning_limit=10
```

tells the simulator to print out 10 warnings per warning category.

warning_limit_dangling

Description

A dangling node, often the result of a design or netlist file problem, is only connected to one device or element (a node in a circuit requires a minimum of two connections). The `warning_limit_dangling` command is used to define the maximum number of listed dangling nodes (default is 50).

Example

Spectre Syntax:

```
usim_opt warning_limit_dangling=100
```

SPICE Syntax:

```
.usim_opt warning_limit_dangling=100
```

tells the simulator to print out 100 dangling nodes.

warning_limit_float

Description

A floating node is an input node (that is, a MOSFET gate) which is not driven by an element or device, and has no DC path to ground. The Virtuoso UltraSim simulator automatically connects floating nodes through a 1e12 ohm resistor (`gmin_float=1e-12`) to ground. The `warning_limit_float` command defines the maximum number of listed floating nodes (default value is 50). The floating nodes are listed in two categories: 1) Nodes connected to MOSFET or JFET gates and 2) nodes not connected to any device gates.

Example

Spectre Syntax:

```
usim_opt warning_limit_float=100
```

SPICE Syntax:

```
.usim_opt warning_limit_float=100
```

tells the simulator to print out 100 floating nodes.

warning_limit_near_float

Description

A nearly floating node is a node with a high resistive path to a driver or ground. A common example is the unconnected substrate of a MOSFET. The `warning_limit_near_float` command defines the maximum number of listed nodes which have a weak DC path to ground (default value is 50). These nodes are listed in two categories: 1) Nodes connected to MOSFET or JFET gates and 2) nodes not connected to any device gates.

Example

Spectre Syntax:

```
usim_opt warning_limit_near_float=100
```

SPICE Syntax:

```
.usim_opt warning_limit_near_float=100
```

tells the simulator to print out 100 nodes with a weak DC path to ground.

warning_limit_ups

Description

Defines the maximum number of listed large resistors in a power net that are detected by the Virtuoso UltraSim power network solver (UPS). The default value is 50.

Example

Spectre Syntax:

```
usim_opt warning_limit_ups=100
```

SPICE Syntax:

```
.usim_opt warning_limit_ups=100
```

tells the simulator to print out 100 large resistors in power net.

warning_node_omit

Spectre Syntax

```
usim_opt warning_node_omit=[node1 node2 …]
```

SPICE Syntax

```
.usim_opt warning_node_omit=[node1 node2 …]
```

Description

Allows you to filter out specific nodes related to dangling, floating, and nearly-floating nodes from warning messages. Wildcards can be used to define these nodes (see "Wildcard Rules" on page 55 for more information).

Examples

In the following Spectre syntax example

```
usim_opt warning_node_omit=[x1.x23.uncon20]
```

tells the Virtuoso UltraSim simulator to exclude the `x1.x2.uncon20` node from the node list of dangling, floating, and near-floating warning messages.

In the following SPICE syntax example

```
.usim_opt warning_node_omit=[x3.x*]
```

tells the simulator to exclude all nodes under the `x3.x*` hierarchy from the node list.

In the next example

```
.usim_opt warning_node_omit=[x1.x23.uncon20 x3.x*.uncon*]
```

tells the simulator to exclude the `x1.x23.uncon20` node and all nodes matching `x3.x*.uncon*` from the node list.

Simulation Start Time Option

sim_start

Description

The Virtuoso UltraSim simulator allows you to start the simulation at a user-defined time using the `sim_start` option.

Table 3-49 sim_start Option

Option	Description
`sim_start`	Simulation starts at the specified time value

Example

Spectre Syntax:

```
usim_opt sim_start=10n
```

SPICE Syntax:

```
.usim_opt sim_start=10n
```

tells the Virtuoso UltraSim simulator to start the simulation at 10 ns.

Simulation Progress Report Control Options

Description

These options are used to print out simulation progress reports to a standard output display device (`stdout`) or log file during transient simulation. If the options are not specified, the Virtuoso UltraSim simulator prints out progress reports at 10% intervals during the transient simulation, or every two hours, whichever occurs first.

progress_t

To define the time interval (in minutes) the simulator prints out the transient simulation progress report to a `stdout` or log file, use

```
progress_t=time
```

Note: Any value for time, other than a whole number, is ignored and the default is used.

progress_p

To define the interval (in transient percentage) the simulator prints out the transient simulation progress report to a `stdout` or log file, use

```
progress_p=percentage
```

Note: This option can also be used to specify the DC progress report.

Examples

In the following Spectre syntax example

```
usim_opt progress_t=5
```

tells the simulator to print out a progress report every 5 minutes.

In the following SPICE syntax example

```
.usim_opt progress_p=2
```

tells the simulator to print out a progress report at the completion of every 2% of transient simulation.

In the next example

```
.usim_opt progress_t=10 progress_p=5
```

tells the simulator to print out progress reports at the completion of every 5% of the transient simulation, or every 10 minutes, whichever occurs first.

Model Building Progress Report

Prior to simulation, generating analog or digital table models for model building usually only takes a few seconds to complete. If model building takes longer, the Virtuoso UltraSim simulator prints a progress report in the log file every five minutes (default). The progress report time interval to print can be adjusted using the `model_progress_t` option.

model_progress_t

Description

The `model_progress_t` option defines the time period the Virtuoso UltraSim simulator uses to print out a progress report during model building (minimum time value is one minute).

Table 3-50 model_progress_t Option

Option	Description
`model_progress_t=`*`value`*	Specifies time period required to print out the model building progress report.

Example

Spectre Syntax:

```
usim_opt model_progress_t=2
```

SPICE Syntax:

```
.usim_opt model_progress_t=2
```

tells the Virtuoso UltraSim simulator to print out model building progress reports every two minutes.

Local Options Report

Spectre Syntax

```
usim_opt block_dump=<0|1|2> [block_depth=<depth_value>]
```

SPICE Syntax

```
.usim_opt block_dump=<0|1|2> [block_depth=<depth_value>]
```

Description

This option allows you to print locally and globally defined simulation options, so you can identify which simulation options are being used for specific blocks in the circuit design. The simulation options are printed to a Virtuoso UltraSim report file (`.usim_opt_rpt`) and also appear as a message in the log file (`.ulog`).

The `.ulog` file contains the following lines which indicate the start and stop time points for the local options:

```
Starting reporting local options in: <filename>
Ending reporting local options
```

The local simulation options are located under the `.usim_opt scope` heading and the global simulation options are located under the `Top Level Options` heading in the report file.

Table 3-51 block_dump and block_depth Options

Option	Description
`block_dump=<0|1|2>`	Defines the report mode. **0** - Report is not generated (default). **1** - Detailed report containing subcircuits and/or instances is generated. If all of the instances for the subcircuit share a common option set, only the subcircuit name is printed. If instances share the same option set as the hierarchy above, the instances are omitted from the report. **2** – Complete report listing all of the instances is generated.
`block_depth=<depth_value>`	Defines the hierarchical depth [optional]. The default value is the maximum hierarchical depth of the circuit design. If `block_depth=0`, only the global option set is printed.

Example

Spectre Syntax:

```
usim_opt block_dump=1
usim_opt sim_mode=ms speed=6
usim_opt sim_mode=da analog=2 speed=4 inst=[X1]
usim_opt sim_mode=s inst=[MNIV1]
```

SPICE Syntax:

```
.usim_opt block_dump=1
.usim_opt sim_mode=ms speed=6
.usim_opt sim_mode=da analog=2 speed=4 inst=[X1]
.usim_opt sim_mode=s inst=[MNIV1]
```

The Virtuoso UltraSim simulator generates the following `<filename>.usim_opt_rpt` file:

```
****************************************************************************
.TITLE 'This file is :./usim.usim_opt_rpt
Options at all the levels are printed
Top Level Options:
The options as follows,
    .usim_opt
   * General Options
    + sim_mode=ms
    + speed=6
    + postl=0
    + pn=1
    + preserve=0
    + analog=1
   * Solver Options
    + tol=20.0000 m
    + method=be
    + trtol=3.5000
    + hier=1
    + maxstep=inf
    ... (continued)
***********************************************************************
.usim_opt scope:
    #mult2x2
The options as follows,
    .usim_opt
   * General Options
    + sim_mode=da
    + speed=4
    + postl=0
    + pn=1
    + preserve=0
    + analog=2
   * Solver Options
    + tol=5.0000 m
    + method=trap
    + trtol=4.5536
    + hier=1
    + maxstep=inf
    ... (continued)
***********************************************************************
.usim_opt scope:
    MNIV1    (n3p3fets)


The options as follows,
    .usim_opt
   * General Options
    + sim_mode=s
    + speed=6
    + postl=0
    + pn=1
    + preserve=0
    + analog=1
```

```
* Solver Options
 + tol=20.0000 m
 + method=gear2
 + trtol=3.5000
 + hier=1
 + maxstep=inf
```

Node Topology Report

Description

The Virtuoso UltraSim simulator `node_topo_report` option allows you to copy node topology analysis results into the following types of ASCII report files:

- Floating node (`.floating_rpt` file extension)
- Nearly floating node (`.weak_floating_rpt`)
- Dangling node (`.dangling_rpt`)

The default setting is `node_topo_report=0`, where node topology report files are not generated by the simulator (initial node topology warnings are still copied into the log files).

Example

Spectre Syntax:

```
usim_opt node_topo_report=1
```

SPICE Syntax:

```
.usim_opt node_topo_report=1
```

tells the Virtuoso UltraSim simulator to generate node topology reports for floating, nearly floating, and dangling nodes. If your netlist file is named `netlist.sp`, the simulator creates `netlist.weak_floating_rpt`, `netlist.floating_rpt`, and `netlist.dangling_rpt` files.

Resolving Floating Nodes

Description

To avoid simulation problems related to floating nodes, the Virtuoso UltraSim simulator automatically inserts a resistor between the floating node and ground. The value of the resistor is defined by `gmin_float`.

gmin_float

Defines the resistor value used for grounding floating nodes (default `gmin_float` value is 1e-12).

Example

Spectre Syntax:

```
usim_opt gmin_float=1e-10
```

SPICE Syntax:

```
.usim_opt gmin_float=1e-10
```

tells the Virtuoso UltraSim simulator to add a 1e10 ohm resistor between any floating node and ground.

Flattening Circuit Hierarchy Option

Because Virtuoso UltraSim is a Fast SPICE simulator, it is able to handle large designs due to its *true hierarchical* approach. The basic idea is to consider subcircuits which are the same and see the same stimuli as one subcircuit. This allows a significant performance improvement compared to flat simulation. There is a certain overhead used for traversing the hierarchy. For circuits where each subcircuit shows different behavior, it can be advantageous to trade memory usage for speed, by flattening the circuit hierarchy.

With the exception of the SPICE and Analog modes, the Virtuoso UltraSim simulator uses an autodetect mode to detect the circuit hierarchy by default. If you want to flatten this circuit, you can use the `hier` command. Even with a flattened netlist file, the Virtuoso UltraSim simulator uses the same simulation engine.

hier

Description

Defines the hierarchy approach the Virtuoso UltraSim simulator applies to the circuit.

Table 3-52 hier Options

Option	Description
`hier=0`	Flattens the netlist file

Table 3-52 hier Options, *continued*

Option	Description
`hier=1`	Autodetect hierarchy (default)

Example

Spectre Syntax:

```
usim_opt hier=0
```

SPICE Syntax:

```
.usim_opt hier=0
```

tells the simulator to flatten the entire circuit.

Device Binning

Devices, which are operated out of the model range they were designed for, can lead to a significant simulation error, as well as to convergence problems. The Virtuoso UltraSim simulator provides an error message if it find such devices. If this problem occurs, and you want to continue the simulation, the option `strict_bin` can be set to use the closest model bin for out-of-range devices.

strict_bin

Description

Defines the model binning approach in the Virtuoso UltraSim simulator.

Table 3-53 strict_bin Options

Option	Description
`strict_bin=1`	The Virtuoso UltraSim simulator gives an error message for devices operating out of model range, and stops the simulation (default).
`strict_bin=0`	The simulator gives a warning message for devices operating out of model range, uses the closest model bin available, and continues the simulation.

Example

Spectre Syntax:

```
usim_opt strict_bin=0
```

SPICE Syntax:

```
.usim_opt strict_bin=0
```

tells the simulator to give a warning and uses the closest model bin for models out of model range.

Element Compaction

By default, the Virtuoso UltraSim simulator compacts parallel elements and replaces them with a newly named element. This approach yields better performance. However, in some cases, this may result in missing current or element probes in the simulation result files. To overcome this limitation, element compaction can be disabled.

elem_compact

Description

Allows to disable element compaction

Table 3-54 elem_compact Options

Option	Description
`elem_compact=1`	Enables element compaction (default)
`elem_compact=0`	Disables element compaction

Example

Spectre Syntax:

```
usim_opt elem_compact=0
```

SPICE Syntax:

```
.usim_opt elem_compact=0
```

tells the simulator to not perform element compaction.

Threshold Voltages for Digital Signal Printing and Measurements

The Virtuoso UltraSim simulator uses logic waveforms for the following statements: `.lprint`/`.lprobe`, `usim_ta`, and `usim_nact`. You can set the threshold voltages by using arguments with each of the aforementioned statements or by defining the threshold voltages using the `vl` and `vh` options. These options can be set globally for the entire circuit or locally for an instance or subcircuit.

Note: Local settings overwrite global settings.

vh

Description

Defines the threshold value for logic `1`. Any signal above this value is considered `1`.

Table 3-55 vh Option

Option	Description
vh=*value*	High threshold voltage (double, unit V). If not specified, the default value is 70% of vdd. If the `vdd` option is not specified, `vh` is defined as 70% of the highest voltage supply in the circuit.

Example

Spectre Syntax:

```
usim_opt vh=2.3
usim_opt vh=1.2 inst=XDIGITAL
```

SPICE Syntax:

```
.usim_opt vh=2.3
.usim_opt vh=1.2 inst=XDIGITAL
```

tells the Virtuoso UltraSim simulator for block `XDIGITAL` to consider signals above 1.2 v to be logic 1, and for all signals outside block `XDIGITAL`, use 2.3 v as the threshold for logic 1.

vl

Description

Defines the threshold value for logic `0`. Any signal below the value is considered `0`.

Table 3-56 vl Option

Option	Description
vl=*value*	Low threshold voltage (double, unit V). If not specified, the default value is 30% of vdd. If the `vdd` option is not specified, `vl` is defined as 30% of the highest voltage supply in the circuit.

Example

Spectre Syntax:

```
usim_opt vl=0.9
```

SPICE Syntax:

```
.usim_opt vl=0.9
```

tells the simulator to print a logic `0` for all signal values below 0.9 v.

Hierarchical Delimiter in Netlist Files

hier_delimiter

Spectre Syntax

```
usim_opt hier_delimiter=”\\”
```

SPICE Syntax

```
.usim_opt hier_delimiter=”\\”
```

Description

The default hierarchical delimiter is a single period (`.`) but can be changed by setting the `hier_delimiter` option.

Notes:

- This option has to be set as the first line in the top level input netlist file.
- To define the delimiter as " or \, the *Escape* symbol is required (for example, `usim_opt hier_delimiter="\""`).

Table 3-57 hier_delimiter Option

Option	Description	Type	Default
`hier_delimiter`	Specifies the hierarchical delimiter in the netlist file	char	.

Example

Spectre Syntax:

```
usim_opt hier_delimiter="%"
```

SPICE Syntax:

```
.usim_opt hier_delimiter="%"
```

hiernode_lookup

Spectre Syntax

```
usim_opt hiernode_lookup=2
```

SPICE Syntax

```
.usim_opt hiernode_lookup=2
```

Description

The Virtuoso UltraSim simulator, by default, does not allow you to use node and element names containing a period (`.`) because this symbol is reserved as a hierarchical delimiter.

In special cases, a period may be used as a hierarchical delimiter and as part of node or element names. You can use `hiernode_lookup=2` to enable the Virtuoso UltraSim simulator to consider the period as part of a node or element name.

For example, a probe or measure statement can be applied to `x0.x1.x2.nd`, where `x0.x1` is the hierarchical instance name and `x2.nd` is the node name. If `hiernode_lookup=2` is used, the Virtuoso UltraSim simulator automatically identifies the hierarchical instance name and reserves `x2.nd` as the node name.

Table 3-58 hiernode_lookup Options

Option	Description
`hiernode_lookup=0`	Period (`.`) cannot be used as part of node or element names (default)
`hiernode_lookup=2`	Period can be used as part of node or element names

MOSFET Gate Leakage Modeling with Verilog-A

Description

Using Verilog-A modules or controlled sources to model gate leakage effects in MOSFET devices may cause conservative partitioning and slow simulation speed. The Virtuoso UltraSim simulator `search_mosg` option allows you to select more aggressive partitioning and a faster simulation speed.

Table 3-59 search_mosg Options

Option	Description
`search_mosg=0`	Search is not performed for MOSFET gate leakage Verilog-A models or controlled sources (default)
`search_mosg=1`	Automatic search is performed for MOSFET gate leakage Verilog-A models and controlled sources

Example

Spectre Syntax:

```
usim_opt search_mosg=1
```

SPICE Syntax:

```
.usim_opt search_mosg=1
```

enables the simulator to automatically search for MOSFET gate leakage Verilog-A models or controlled sources using more aggressive partitioning, resulting in a faster simulation speed.

Automatic Detection of Parasitic Bipolar Transistors

Description

Circuit designers often want to simulate the effects of parasitic bipolar junction transistor (BJT) devices formed in the triple well CMOS process. Including these transistors in the simulation may result in conservative partitioning and slow simulation speed. The `parasitic_bjt` option allows you to control the way the Virtuoso UltraSim simulator handles the parasitic BJT devices, resulting in much faster simulation speed.

Note: The simulator can only detect parasitic vertical PNP BJTs with the emitter connected to the body of a NMOSFET.

Table 3-60 Parasitic BJT Options

Option	Description
`parasitic_bjt=0`	No detection of parasitic BJT devices (default)
`parasitic_bjt=1`	Detect parasitic vertical PNP BJT devices and invoke aggressive partitioning
`parasitic_bjt=2`	Detect and remove parasitic vertical PNP BJT devices

Examples

In the following Spectre syntax example

```
usim_opt parasitic_bjt=1
```

tells the Virtuoso UltraSim simulator to detect parasitic vertical PNP BJT devices and to invoke aggressive partitioning.

In the following SPICE syntax example

```
.usim_opt parasitic_bjt=2
```

tells the simulator to cut away all the parasitic vertical PNP BJT devices.

Duplicate Subcircuit Handling

duplicate_subckt

Description

You can define multiple definitions for a subcircuit using the `duplicate_subckt` option in the netlist file.

Example

Setting `duplicate_subckt` to

```
.usim_opt duplicate_subckt=1
```

uses the last definition of the subcircuit and overrides all the previous subcircuit definitions. If `duplicate_subckt` is set to zero (0), the Virtuoso UltraSim simulator stops the simulation, and displays an error message if duplicate subcircuits are detected.

Bus Signal Notation

buschar

Spectre Syntax

```
usim_opt buschar="<>"
```

SPICE Syntax

```
.usim_opt buschar="<>"
```

Description

The Virtuoso UltraSim simulator resolves bus signals into individual signals when reading Verilog netlist files (`.vlog`). The `buschar` option and either `<>` or `[]` is used to set the bus notation. The exception is bus notation for vector and vcd stimuli which is set using vector and vcd options (see Chapter 13, “Digital Vector File Format” and Chapter 14, “Verilog Value Change Dump Stimuli” for more information).

Table 3-61 buschar Options

Option	Description
`buschar="[]"`	Used as bus notation (default)
`buschar="<>"`	Used as bus notation

Example

Spectre Syntax:

```
usim_opt buschar="<>"
```

SPICE Syntax:

```
.usim_opt buschar="<>"
```

tells the Virtuoso UltraSim simulator to use `<>` as the notation for bus signals read from the structural Verilog netlist file.

Bus Node Mapping for Verilog Netlist File

vlog_buschar

Description

The Virtuoso UltraSim simulator can support name mapping for bus nodes when instantiating analog cells in a structural Verilog netlist file. Bus node mapping between the structural Verilog netlist file and analog cell is based on the order of the nodes. To resolve bus signals into individual signals in the analog netlist file, the `vlog_buschar` option is used to set the bus notation.

Note: The ports of the bus node in the analog cell definition must be continuous because the simulator ends the bus node definition once another node name is encountered.

```
.usim_opt vlog_buschar="front_bus_symbol*end_bus_symbol"
```

where asterisk (`*`) is a keyword and the default bus symbol is `[]` (square brackets). The `front_bus_symbol` and `end_bus_symbol` arguments define the starting and ending letters of bus notation, respectively.

Examples

For the first example

Structural Verilog netlist file:

```
add4 u1 ( .a ({ net1, net2, a1, a2 }), .sum ({ sum1, sum0 }), .vdd(VDD3),
.vss(VSS_DIG) );
```

Analog netlist file:

```
.usim_opt vlog_buschar="_*"
.subckt add4 a_3 a_2 a_1 a_0 vdd vss sum_1 sum_0
```

tells the Virtuoso UltraSim simulator to map the `net1`, `net2`, `a1`, and `a2` nodes in the Verilog netlist file to `a_3`, `a_2`, `a_1`, and `a_0` in the analog netlist file, as well as `sum1` and `sum0` to `sum_1` and `sum_0`.

In the next example

Structural Verilog netlist file:

```
reg [2:0] n;
ram i0 .a (n);
```

Analog netlist file:

```
.usim_opt vlog_buschar="<*>"
subckt ram a<2> a<1> a<0>
```

tells the simulator to map the `n[2]`, `n[1]`, and `n[0]` nodes in the Verilog netlist file to `a<2>`, `a<1>`, and `a<0>` in the analog netlist file.

In the next example

Structural Verilog netlist file:

```
reg [7:0] Addr;
ram i0 ( .A(Addr), vdd(VDD3) );
```

Analog netlist file:

```
.usim_opt vlog_buschar="*"
subckt ram A7 A6 A5 vdd A4 A3 A2 A1 A0
```

tells the simulator that the `A4` through `A0` analog signals are not recognizable as bus nodes because the `vdd` node ends the bus node definition.

Structural Verilog Dummy Node Connectivity

vlog_supply_conn

Spectre Syntax

```
usim_opt vlog_supply_conn=[portname1 node1 portname2 node2 ...]
```

SPICE Syntax

```
.usim_opt vlog_supply_conn=[portname1 node1 portname2 node2 ...]
```

When invoking an analog cell using SPICE syntax from a structural Verilog instance, the redundant ports of the SPICE cell are connected to dummy nodes if the Verilog instance has fewer ports than the SPICE cell. If the power nets (for example, `vdd` and `vss`) are only defined in the SPICE subcircuit, and not the Verilog instance, they are also connected to dummy nodes.

Note: For the Virtuoso UltraSim simulator, the local node always overwrites the global node.

For example, in the Spectre netlist file

```
subckt add a1 a2 sum vdd vss
```

SPICE netlist file:

```
.subckt add a1 a2 sum vdd vss
```

Structural Verilog netlist file:

```
add u1 a1 a2 sum
```

The `vdd` and `vss` nodes cannot be connected to the power net, even if they are declared global nodes in the netlist file.

The `vlog_supply_conn` option is used to connect to

- The structural Verilog dummy node
- Most of the supply node
- The global or internal node of the Verilog instance

Description

This option is used to connect a dummy node to either the global or internal node of the Verilog instance. The port names in the analog netlist file are specified using *`portname1`*, *`portname2`*, ... and *`node1`*, *`node2`*, ... is used to specify the internal or global node names of the Verilog instance.

The option can be applied locally to a Verilog module or instance and is set the same as the other Virtuoso UltraSim simulator local options (see Examples below for more information). The Verilog module and instance can be regarded as a subcircuit and instance when setting the local options. This option is also valid for lower hierarchical level instances.

Note: The `vlog_supply_conn` option has no effect when the port of a subcircuit in the analog netlist file is not a dummy node, and the port is connected to a signal in the Verilog netlist file.

Examples

In the following Verilog netlist file example

```
add u1 a1 a2 sum
```

In the analog netlist file

```
.global global_vdd global_vss
.usim_opt vlog_supply_conn=[vdd global_vdd vss global_vss]
.subckt add a1 a2 sum vdd vss
M1 mid1 a1 vss vss nmos
```

The `global_vdd` node is connected to `vdd` in the `add` subcircuit of instance `u1`, and the `global_vss` node is connected to `vss`.

In the next Verilog netlist file example

```
add u1 a1 a2 sum vdd vss
```

tells the Virtuoso UltraSim simulator that the analog netlist file is the same as the one used in the previous example. The `vdd` and `vss` nodes of the Verilog instance `u1` are not dummy nodes, so they are not connected to the global node by the simulator.

In the following Verilog netlist file local option example

```
xor u1 local_vss in2 out
nand4 u2 in1 in2 in3 in4 out
```

In the analog netlist file

```
.usim_opt vlog_supply_conn=[vdd global_vdd vss local_vss] xdigital.verilog.u1
```

```
.global global_vdd global_vss
.subckt xor in1 in2 out vdd vss
....
.subckt nand4 in1 in2 in3 in4 out vdd vss
```

tells the simulator `vdd` and `vss` of `nand4` for Verilog instance `u2` are still connected to dummy nodes because the option is local and is only applied to Verilog instance `u1`. Also, `vdd` and `vss` of `xor` for instance `u1` are connected to the `global_vdd` (global) and `local_vss` (local) nodes in the Verilog netlist file, respectively.

skip Option

Description

Use the `skip` option to disable a circuit block simulation.

Table 3-62 skip Options

Option	Description
`skip=0`	The Virtuoso UltraSim simulator includes the circuit block in the simulation (default).
`skip=1`	The simulator disables the simulation for the specified circuit blocks. The loading effect of the disabled blocks is considered. The inputs of the remaining circuit, connected to the disabled blocks, are connected to the ground through high resistance (that is, treated as floating nodes).

Examples

In the following Spectre syntax example

```
usim_opt skip=1 inst=x0.x1
```

tells the Virtuoso UltraSim simulator to disable the `x0.x1` circuit block simulation.

In the following SPICE syntax example

```
.usim_opt skip=1 subckt=op_amp
```

tells the simulator to disable the simulation for all instances of the `op_amp` subcircuit.

probe_preserve Option

Description

Use to control the preserve setting of `.probe` statements. If `probe_preserve` is set to `all`, the simulator applies `preserve=all` to all `.probe` statements.

Note: When set to `all`, the `probe_preserve` option overrides `preserve=none|all` specified in `.probe` statements. It does not override `preserve=port`.

Table 3-63 probe_preserve Options

Option	Description
`probe_preserve=none`	Does not have any impact on `.probe` statements (default).
`probe_preserve=all`	Applies `preserve=all` to all `.probe` statements.

Examples

In the following Spectre syntax example

```
usim_opt probe_preserve=all
```

tells the Virtuoso UltraSim simulator to apply `preserve=all` to all `.probe` statements in the netlist.

In the following Spectre syntax example

```
.usim_opt probe_preserve=all
```

tells the Virtuoso UltraSim simulator to apply `preserve=all` to all `.probe` statements in the netlist.

Print File Options

nodecut_file

Description

Enables the Virtuoso UltraSim simulator to print all nodes cut during post-processing into a file (file extension is `nodecut`).

Table 3-64 nodecut_file Options

Option	Description
`nodecut_file=0`	The Virtuoso UltraSim simulator does not print cut nodes into a file (default)
`nodecut_file=1`	The simulator prints all cut nodes into a `.nodecut` file

elemcut_file

Description

Enables the simulator to print all elements cut when thresholds are exceeded into a file (file extension is `elemcut`).

Table 3-65 elemcut_file Options

Option	Description
`elemcut_file=0`	The simulator does not print cut elements into a file (default)
`elemcut_file=1`	The simulator prints all cut elements into a `.elemcut` file

Controlling Text Wrapping of Circuit Check Reports

pcheck_wrap

Description

Controls text wrapping in circuit check reports created by Virtuoso Ultrasim Simulator.

Table 3-66 pcheck_wrap Options

Option	Description
`pcheck_wrap=0`	Disables text wrapping in circuit check reports (defualt).
`pcheck_wrap=1`	Enables text wrapping in circuit check reports.

Changing Resistor, Capacitor, or MOSFET Device Values

.usim_trim

Spectre Syntax

```
usim_trim instance=resistor_name value=resistor_value
usim_trim instance=capacitor_name value=capacitor_value
usim_trim instance=instance_name [w=value] [l=value] [delvto=value]
```

SPICE Syntax

```
.usim_trim instance=resistor_name value=resistor_value
.usim_trim instance=capacitor_name value=capacitor_value
.usim_trim instance=instance_name [w=value] [l=value] [delvto=value]
```

Description

The `usim_trim` option can be used to change the values of resistors and capacitors, and the length, width, and threshold voltage of a MOSFET device, without modifying the netlist file.

Notes:

- The instance name must contain the full hierarchical path
- The `usim_trim` option does not work with stitched `dspf` or `spef` flows
- Only MOSFET device lengths, widths, and threshold voltages can be changed (use the `delvto` device model parameter to adjust threshold voltages)

Examples

In the following Spectre syntax example

```
usim_trim instance=x1.x2.cap5 value=3f
```

tells the Virtuoso UltraSim simulator to change the value of instance `x1.x2.cap5` to 3f (Fahrenheit).

In the following SPICE syntax example

```
.usim_trim instance=x1.x2.x3.res5 value=1k
```

tells the simulator to change the value of instance `x1.x2.x3.res5` to 1k (ohms).

In the next example

```
.usim_trim instance=x1.x2.mp00 w=10e-5 l=5e-5
```

tells the simulator to change the width of instance `x1.x2.mp00` to 10e-5 meters and length to 5e-5 meters.

In the next example

```
.usim_trim instance=x1.mn1 w=1.0e-6
```

tells the simulator to change the width of instance `x1.mn1` to 1.e-6 meters.

.reconnect

Spectre Syntax

Spectre syntax is not supported.

SPICE Syntax

```
.reconnect instport=instance_port_name node=node_name
.reconnect subcktport=subckt_port_name node=node_name
```

Changes the connection of certain instances' ports. This command is useful in the early stage of power network development. During this stage, only the estimates of the power network parasitics are available, and you are required to disable the original connection and establish new connections without manually changing the original netlist.

Table 3-67 .reconnect Option

Option	Description
`instport`	Specifies the port name of the instance whose original connection is to be disconnected. The syntax to define a port of an instance is `instancename.portname` where . is the hierarchical delimiter. For example, the port `vdd` of instance `X1` is defined as `X1.vdd`. The port can be defined explicitly or implicitly in a global statement.
`node`	Specifies the node to which the new connection is to be established. Hierarchical node name is required. Wildcard is not supported.
`subcktport`	Specifies the port name of the subckt whose original connection is to be disconnected. This is applicable to all the instances of the subckt. As in the case of `instport`, the port can be defined explicitly or implicitly. The syntax to specify the port `vdd` of all the instances of subckt `inv` is `inv/vdd` where / is the hierarchical delimiter.

Multiple .reconnect statements are supported. If duplicate specifications are used for the same port of the same instances, the last specification is given priority. If neither the specified port nor the specified node is found in the circuit, UltraSim issues a warning and ignores the command.

Examples

```
.reconnect instport=x1.p node=vcc
```

Suppose that `p` is an explicit port of `x1` and the hierarchical delimiter is ., this command disconnects the original connection of port `p` of `x1` and reconnects a top-level node `vcc` to instance `x1`'s port `p`.

```
.reconnect instport=x1/vdd node=vcc
```

Suppose that `vdd` is defined as a global node and / is the hierarchical delimiter, this command reconnects a top-level node `vcc` to the global node `vdd` inside instance `x1`, including all the hierarchies inside `x1`. The connection of `vdd` in other blocks remains unchanged.

```
.reconnect subcktport=pump.out node=vcc
```

Suppose that `out` is an explicit port of subckt `pump` and it has three instances: `xpump1`, `xpump2`, and `xpump3`, this command tells UltraSim that all instances' port `out`, that is, `xpump1.out`, `xpump2.out`, `xpump3.out`, have to be disconnected from their original connections and reconnected with `vcc`.

```
.reconnect instport=x1.x2.p node=x3.x4.netA
```

Suppose that port `p` is an explicit port, this command tells UltraSim to reconnect node `x3.x4.netA` with the port `p` of instance `x1.x2` and disconnect the original connection of `x1.x2.p`.

Simulator Options: Default Values

The default values for the Virtuoso UltraSim simulator options are listed below in Table 3-68 on page 237. The majority of these options are listed in the *Simulation Options* section of the output log file. The default values may vary for different versions of the simulator.

Table 3-68 Simulator Options and Default Values

Option	Default Value
General	
`sim_mode`	`ms`
`speed`	5
`postl`	0
`analog`	1(`df`/`da`/`ms`) ; ignored(`a`/`s`)
`preserve`	1
`pn`	0
Solver	
`tol`	10 m
`method`	be(`df`/`da`/`ms`); gear2(`a`/`s`)
`trtol`	3.5
`hier`	1(`df`/`da`/`ms`); 0(`a`/`s`)
`maxstep_window`	inf
Device Model	
`mos_method`	df(`df`), da(`da`), a(`ms`/`a`), s(`s`)
`mosd_method`	df(`df`/`da`), a(`ms`/`a`); ignored(`s`)
`diode_method`	Juncap: a(`df`/`da`/`ms`/`a`) s(`s`); other diodes: s(`df`/`da`/`ms`/`a`/`s`)
`vdd`	max. supply voltage
`deg_mod`	r
Post-Layout	
`rshort`	1u ohm
`rvshort`	1u ohm

Table 3-68 Simulator Options and Default Values, *continued*

Option	Default Value
lshort	0 H
lvshort	0 H
minr	0
cgnd	10 zF
cgndr	0
canalog	100 fF
canalogr	450 m
rcr_fmax	1 GHz
DC	
dc	1(df/da/ms); 3(a/s)
dc_exit	0
dc_prolong	0
Simulation	
abstolv	1 uV
abstoli	1 pA
progress_t	120 min
progress_p	10%
vl	supply voltage * 0.3
vh	supply voltage * 0.7
sim_start	0
dump_step	0
gmin_allnodes	0
cmin_allnodes	0
Environment	
ade	0
Parser	
hier_delimiter	. (period)

Table 3-68 Simulator Options and Default Values, *continued*

Option	Default Value
duplicate_subckt	0
warning_limit	5
warning_limit_dangling	50
warning_limit_float	50
warning_limit_near_float	50
warning_limit_ups	50
Model	
strict_bin	1
Database	
buschar	0
Output	
wf_format	SST2
wf_maxsize	inf
wf_reltol	5 m(dt/da); 100 n(a/ms)
wf_tres	100 f
wf_abstolv	1 uV
wf_abstoli	1 p
wf_filter	2(df/da/ms); 1(a); 0(s)
pa_elemlen	20
Power Net Solver	
pn_max_res	1000 ohm

Notes

- The `df` abbreviation stands for global `df` mode, `da` for global `da` mode, `ms` for global `ms` mode, `a` for global `a` mode, and `s` for global `s` mode.

- The Virtuoso UltraSim simulator automatically promotes `analog` from `1` to `2` when simulating a small design (that is, a design with less than 200 active devices). The `analog` option is ignored in global `a/s` mode.
- For more information about the Virtuoso UltraSim simulation option definitions, refer to the option descriptions in this chapter or use the `-help .usim_opt` command.

4

Post-Layout Simulation Options

This chapter describes the Virtuoso® UltraSim™ simulator post-layout simulation options.

Parasitic effects (for example, wire delays and coupling) are becoming increasingly important factors in integrated circuit design. The Virtuoso UltraSim post-layout simulator considers these effects, modeling parasitic resistors and capacitors extracted from the layout environment.

The Virtuoso UltraSim simulator is designed to handle the demands of most of the post-layout simulation flows (refer to Figure 4-1 on page 242 for more information) and supports the backannotation of parasitic resistors and capacitors (RCs) from a detailed standard parasitic format (DSPF) file, a standard parasitic exchange format (SPEF) file, parasitic capacitance from a node capacitance file, and extracted device layout parameters from a DPF file.

The most important challenges for post-layout simulation is processing large numbers of parasitic elements and supporting a variety of layout parasitic extraction flows. The Virtuoso UltraSim simulator uses advanced RC reduction techniques to handle large numbers of parasitic elements while preserving circuit timing domain behavior. There are three general post-layout simulation methodologies:

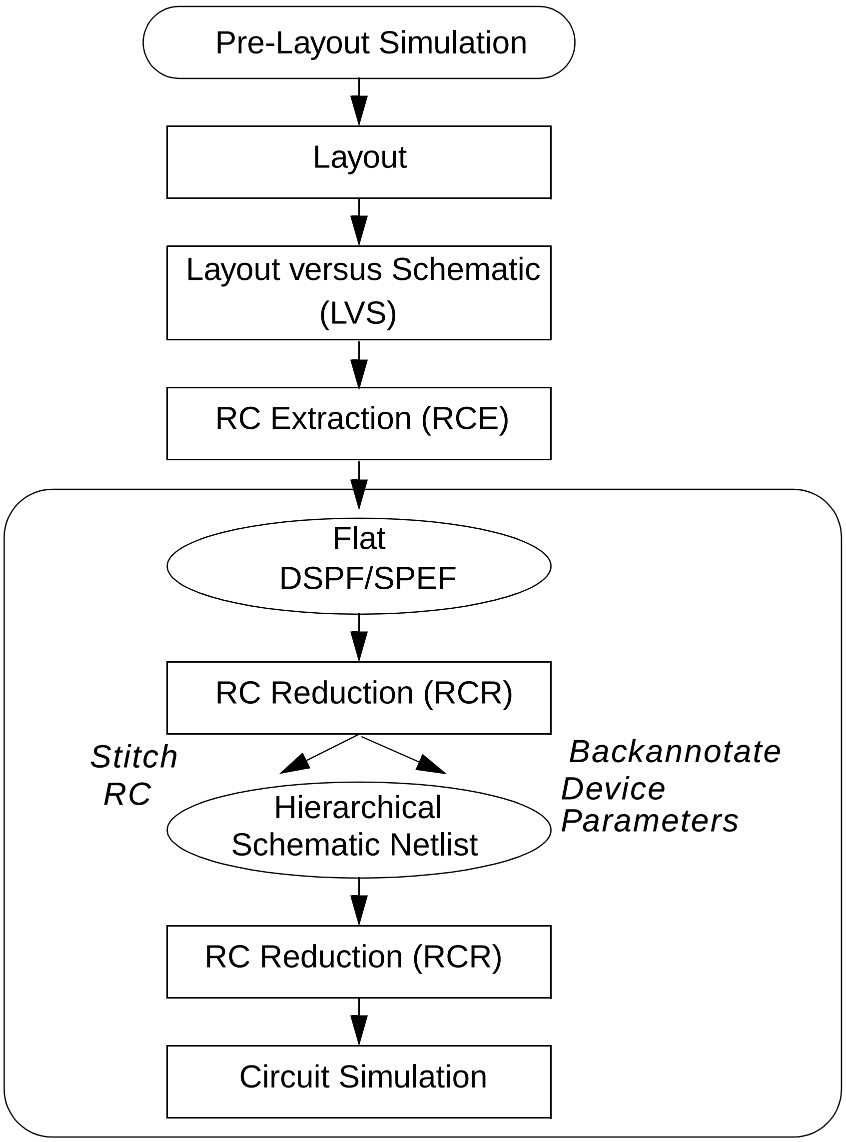
1. **Flat RC netlist file** is a simple approach, allowing you to use a large number of elements and devices in the netlist, but it tends to be more memory and time consuming. An example of a flat RC netlist is a flat DSPF file.

2. **Hierarchical RC netlist file** is an approach in which the Virtuoso UltraSim simulator preserves the hierarchical structure of the netlist file to reduce run time and memory requirements. This simulation method is generally faster than the flat RC netlist file approach. The main drawback is that the parasitic information is embedded inside the netlist file, making it more difficult to extract the information.

3. **Backannotation and stitching of parasitic files** is an approach that has the simulator preserve the design hierarchy to reduce run time and memory requirements, and accepts mixed post-layout netlist file formats to support simulation of circuit designs. For example, designs in which some blocks have successfully passed layout versus schematic (LVS) verification and have post-layout netlist files, whereas other blocks

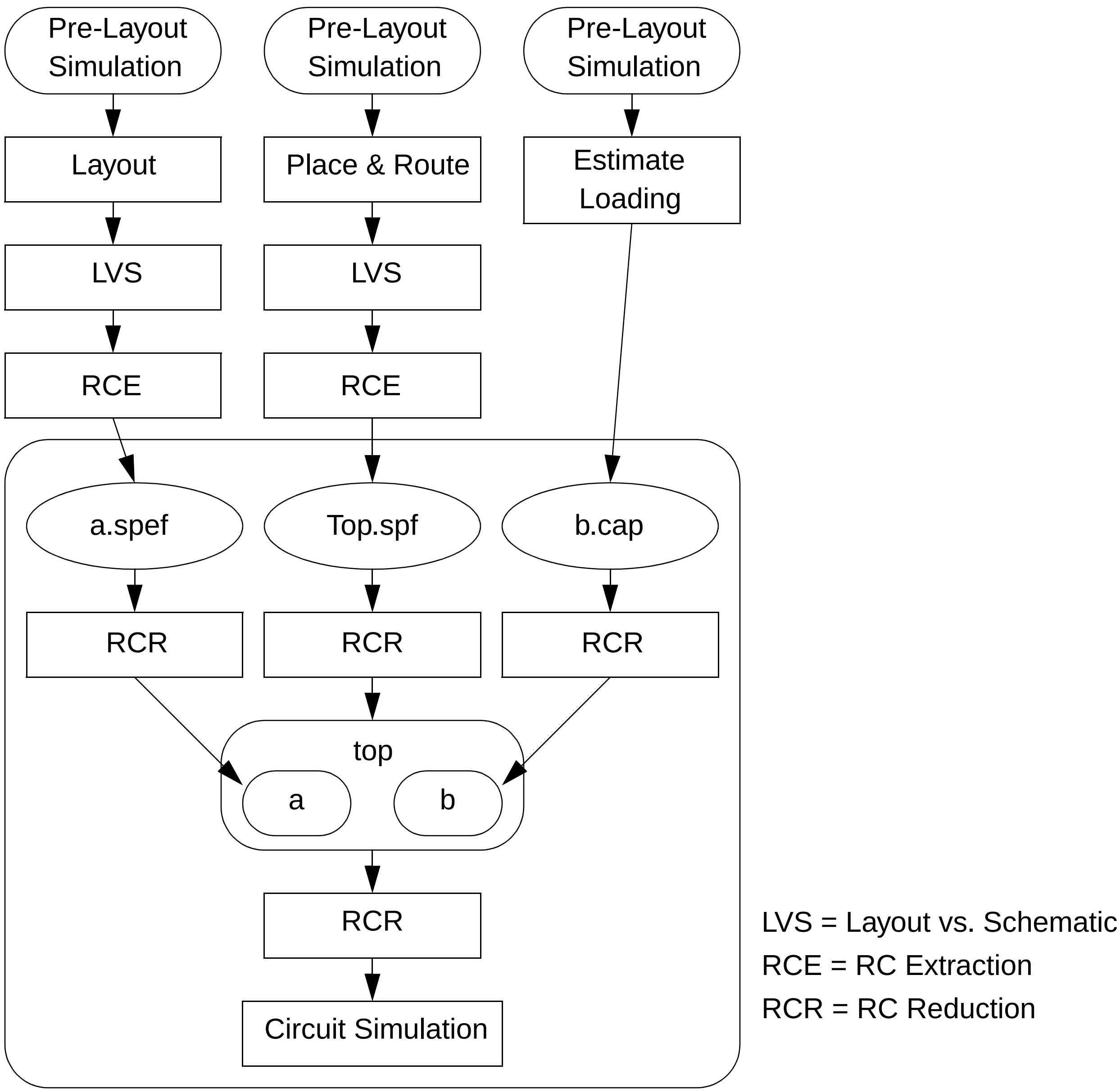
remain at an early stage of development and only have estimated capacitive loading. See Figure 4-2 on page 243 to view this full-chip simulation flow example graphically.

Figure 4-1 Post-Layout Simulation Flow



You can use the Virtuoso UltraSim simulator backannotation flow to start full chip simulation early in the design cycle, and to gradually replace pre-layout views with post-layout netlist files, as design blocks pass layout and extraction verification. “What if” analyses can also be performed by overwriting the extracted capacitance on critical nets. The backannotation method preserves schematic device and net names to facilitate cross probing and circuit debugging. The simulator post-layout flows can be integrated into top-down or bottom-up design flows in which extractions are performed at various levels of hierarchy, following a graybox methodology.

Figure 4-2 Full-Chip Simulation Flow



The Virtuoso UltraSim simulator also provides a set of options which allow selective parasitic backannotation based on different values, such as RC elements, hierarchy level, and net names.

This chapter describes in detail the Virtuoso UltraSim simulator RC reduction method that allows you to significantly reduce the number of parasitic elements while maintaining simulation accuracy. The stitching of parasitic resistor (R) and capacitor (C) elements, and the backannotation of layout device parameters from parasitic files into hierarchical schematic or netlist files, is also described.

RC Reduction Options

The Virtuoso UltraSim simulator post-layout simulation options allow you to short small resistors, ground small coupling capacitors, and perform RC reduction in order to speed up simulation, while preserving the time domain behavior of the circuits. All the RC reduction related options can be applied globally and locally.

The simulator also provides the high-level `postl` option which you can use to control the trade-off between simulation accuracy and performance. Cadence recommends that you read more about "postl" on page 253 before starting a post-layout simulation.

The Virtuoso UltraSim simulator supports the following RC reduction options:

ccut

Spectre Syntax

```
usim_opt ccut=value
```

SPICE Syntax

```
.usim_opt ccut=value
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.usim_opt ccut`).

Description

This option allows you to cut capacitors, depending on the value of the capacitors. Capacitors less than `ccut` are cut (that is, open circuited) during parsing. The `ccut` option is helpful with large post-layout netlist files containing small capacitors. To reduce memory consumption, `ccut` needs to be defined at the beginning of the netlist file, so cutting is performed during parsing. The default is 0.

Example

Spectre Syntax:

```
usim_opt ccut=0.1ff
```

SPICE Syntax:

```
.usim_opt ccut=0.1ff
```

tells the Virtuoso Ultrasim simulator to cut all capacitors with a value < 0.1 ff during parsing.

cgnd

Spectre Syntax

```
usim_opt cgnd=value
```

SPICE Syntax

```
.usim_opt cgnd=value
```

Description

This `cgnd` option defines the absolute threshold value for grounding coupling capacitors. A capacitor is considered a coupling capacitor if neither of its terminals are connected to a ground voltage source. A ground voltage source is one that has a path to ground directly or through other voltage sources. Coupling capacitors can lead to significantly longer run times and may not contribute appreciably to simulation accuracy. A coupling capacitor can be split into two grounded capacitors with the same capacitance value: One at each terminal of the original capacitor.

With this option, a coupling capacitor is grounded if its value is less than `cgnd`. The grounding of coupling capacitors is automatically enabled by setting the high-level post-layout control parameter `postl`. The default value depends on the setting for `postl`. See Table 4-1 on page 253 for more details.

Example

Spectre Syntax:

```
usim_opt cgnd=1pf
```

SPICE Syntax:

```
.usim_opt cgnd=1pf
```

tells the Virtuoso UltraSim simulator to ground all coupling capacitors with a value < 1 pf.

cgndr

Spectre Syntax

```
usim_opt cgndr=value
```

SPICE Syntax

```
.usim_opt cgndr=value
```

Description

The `cgndr` option defines the relative threshold value for grounding coupling capacitors. A coupling capacitor is grounded if the ratio of its value to the total node capacitance on both sides is less than `cgndr`. The range of `cgndr` is between 0 and 1. The default value depends on the setting for `postl`. See Table 4-1 on page 253 for more details.

Example

Spectre Syntax:

```
usim_opt cgndr=0.01
```

SPICE Syntax:

```
.usim_opt cgndr=0.01
```

tells the Virtuoso UltraSim simulator to ground any coupling capacitor if the ratio of its value to the total node capacitance on both sides of the capacitor is < 0.01.

rcr_fmax

Spectre Syntax

```
usim_opt rcr_fmax=value
```

SPICE Syntax

```
.usim_opt rcr_fmax=value
```

Description

The `rcr_fmax` option defines the maximum frequency of interest for RC reduction (default is 1.0 GHz.). If the chosen value for `rcr_fmax` is less than the maximum operating frequency of interest, you may experience accuracy loss for frequencies higher than the specified `rcr_fmax` value.

Example

Spectre Syntax:

```
usim_opt rcr_fmax=10G
```

SPICE Syntax:

```
.usim_opt rcr_fmax=10G
```

tells the Virtuoso UltraSim simulator to adjust the RCR reduction accordingly (suitable for designs up to 10 GHz for the frequency of interest).

rcut

Spectre Syntax

```
usim_opt rcut=value
```

SPICE Syntax

```
.usim_opt rcut=value
```

Description

This option can be used to cut resistors with values larger than the specified `rcut` value (that is, open-circuit resistors). The default is 1e12 ohm.

Example

Spectre Syntax:

```
usim_opt rcut=1e14
```

SPICE Syntax:

```
.usim_opt rcut=1e14
```

tells the Virtuoso UltraSim simulator to cut all resistors with a value > 1e14 ohm.

rshort

Spectre Syntax

```
usim_opt rshort=value
```

SPICE Syntax

```
.usim_opt rshort=value
```

Description

Signal net resistors with a value less than `rshort` are short-circuited. The default value depends on the setting for `postl`. See Table 4-1 on page 253 for more details.

Example

Spectre Syntax:

```
usim_opt rshort=1 subckt=AMP
```

SPICE Syntax:

```
.usim_opt rshort=1 subckt=AMP
```

tells the Virtuoso Ultrasim simulator to short-circuit all signal net resistors < 1 ohm in all instances of subcircuit `AMP`.

rvshort

Spectre Syntax

```
usim_opt rvshort=value
```

SPICE Syntax

```
.usim_opt rvshort=value
```

Description

For any resistor connected to an independent voltage source, if its value is less than `rvshort`, the resistor is short-circuited. The default value depends on the setting for `postl`. See Table 4-1 on page 253 for more details.

Example

Spectre Syntax:

```
usim_opt rvshort=1 subckt=SUPPLY
```

SPICE Syntax:

```
.usim_opt rvshort=1 subckt=SUPPLY
```

tells the Virtuoso UltraSim simulator to short-circuit all rail resistors with a value less than 1 ohm in all instances of subcircuit `SUPPLY`.

postl

Spectre Syntax

```
usim_opt postl=0|1|2|3|4
```

SPICE Syntax

```
.usim_opt postl=0|1|2|3|4
```

Description

This is a high-level simulator option that allows you to control the trade-off between simulation accuracy and performance.

postl=0 is designated for simulation of a pre-layout netlist file containing a few resistors and capacitors. The Virtuoso UltraSim simulator does not perform RC filtering or reduction, except shorting extremely small resistors, and grounding extremely small capacitors to allow stable simulation.

postl=1, **postl=2**, and **postl=3** are intended for post-layout simulation. As the `postl` level is raised, the Virtuoso UltraSim simulator applies more aggressive RC reduction. As a result, run time and memory usage are reduced at the cost of slightly degraded simulation accuracy.

For **postl=4**, most of the resistors in signal nets are eliminated and most coupling capacitors are grounded. This produces a post-layout simulation where only grounded parasitic capacitance is taken into account.

Table 4-1 Virtuoso UltraSim Post-Layout Options

	Pre-Layout	Post-Layout	Post-Layout	Post-Layout	Post-Layout
`postl`	0 (default)	1	2	3	4
`RC reduction`	no	yes	yes	yes	no
`rcr_fmax (GHz)`	1.0	1.0	1.0	1.0	1.0
`rshort (W)`	1e-6	1e-3	0.01	0.1	100
`rvshort (W)`	1e-6	1e-3	0.01	0.1	100
`cgnd (F)`	1e-20	1e-16	1e-15	1e-14	1e-14

Table 4-1 Virtuoso UltraSim Post-Layout Options, *continued*

	Pre-Layout	Post-Layout	Post-Layout	Post-Layout	Post-Layout
`cgndr`	0	0.01	0.1	0.1	0.3

Example

Spectre Syntax:

```
usim_opt postl=1 rshort=1 subckt=VCO
```

SPICE Syntax:

```
.usim_opt postl=1 rshort=1 subckt=VCO
```

The Virtuoso UltraSim simulator applies `postl=1` to all instances of subcircuit `VCO`. Any resistors connected to signal nets in all instances, if the values of the resistors are less than 1 ohm, are short-circuited. Default values are used for all other options associated with `postl`, such as `rcr_fmax`, `rvshort`, `cgnd`, and `cgndr`.

Excluding Resistors and Capacitors from RC Reduction

preserve

Spectre Syntax

```
usim_opt preserve=1 inst=[res1 res2 cap1 cap2] model=[model1 model2..]
     preserve_file=[“filename1” “filename2”...]
```

SPICE Syntax

```
.usim_opt preserve=1 inst=[res1 res2 cap1 cap2] model=[model1 model2..]
     preserve_file=[“filename1” “filename2”...]
```

Description

The `preserve=1` command is used to exclude resistors and capacitors from RC reduction.

Table 4-2 preserve=1 Options

Option	Description
`res1`, `res2`	Specifies the resistors to be excluded from RC reduction (must use full hierarchical name).
`cap1`, `cap2`	Specifies the capacitors to be excluded from RC reduction (must use full hierarchical name).
`filename1, filename2...`	The name of the files which contain the resistors and capacitors to be excluded (full hierarchical name for resistors and capacitors needs to be included in files).
`model1`, `model2`	Specifies the model names. The model names need to use the resistor or capacitor model names (all instances of the model specified are excluded from RC reduction).

If all of the options are used in the statement, the Virtuoso UltraSim simulator only uses the resistors and capacitors shared between the specified options (that is, RC reduction options, such as `postl`, `rshort`, and `rvshort` are applied only to the resistors and capacitors not specified in this option).

Example

Spectre Syntax:

```
usim_opt preserve=1 inst=x1.x3.r1 preserve_file=[“inst.txt”]
```

SPICE Syntax:

```
.usim_opt preserve=1 inst=x1.x3.r1 preserve_file=[“inst.txt”]
```

The `inst.txt` file contains the following resistors:

```
X3.x2.res1
X2.res3
R1
```

tells the Virtuoso UltraSim simulator to exclude the `x1.x3.r1`, `X3.x2.res1`, `X2.res3`, and `R1` resistors from RC reduction.

Stitching Files

capfile

Spectre Syntax

```
usim_opt capfile="<instance|subckt> file"
```

SPICE Syntax

```
.usim_opt capfile="<instance|subckt> file"
```

Description

The `capfile` option specifies how to load a cap file into the Virtuoso UltraSim simulator.

Arguments

`path`	The full hierarchical path of the instance for which the cap file is prepared. Wildcards are supported (for more information about wildcards, see "Wildcard Rules" on page 55). You can also specify the subcircuit name as the path. All instances of the specified subcircuit are stitched. If the path is not specified, and the cap file contains a `.subckt` statement, all instances with the same subcircuit name are stitched (or the simulator assumes that the cap file is prepared for the entire design, and the quotation marks can be omitted).
`file`	The name of the parasitic file. The Virtuoso UltraSim simulator can read compressed parasitic files in `gz` format (files need to have the `.gz` extension).

For more information about parsing options for cap files, refer to "Parsing Options for Parasitic Files" on page 263.

Example

For the design shown in Figure 4-2 on page 243, the parasitic files need to be specified as follows (Spectre syntax example):

```
usim_opt spef="a a.spef" spf=”Top.spf.gz” capfile="b b.cap"
```

Note: You need to make sure the parasitic elements specified in the parasitic files do not overlap and that `Top.spf` is extracted until the level for the `a` and `b` instances is reached (that is, the file contains all elements inside `top`, excluding the elements in `a` and `b`).

dpf

Spectre Syntax

```
usim_opt dpf=”<instance|subckt> file”
```

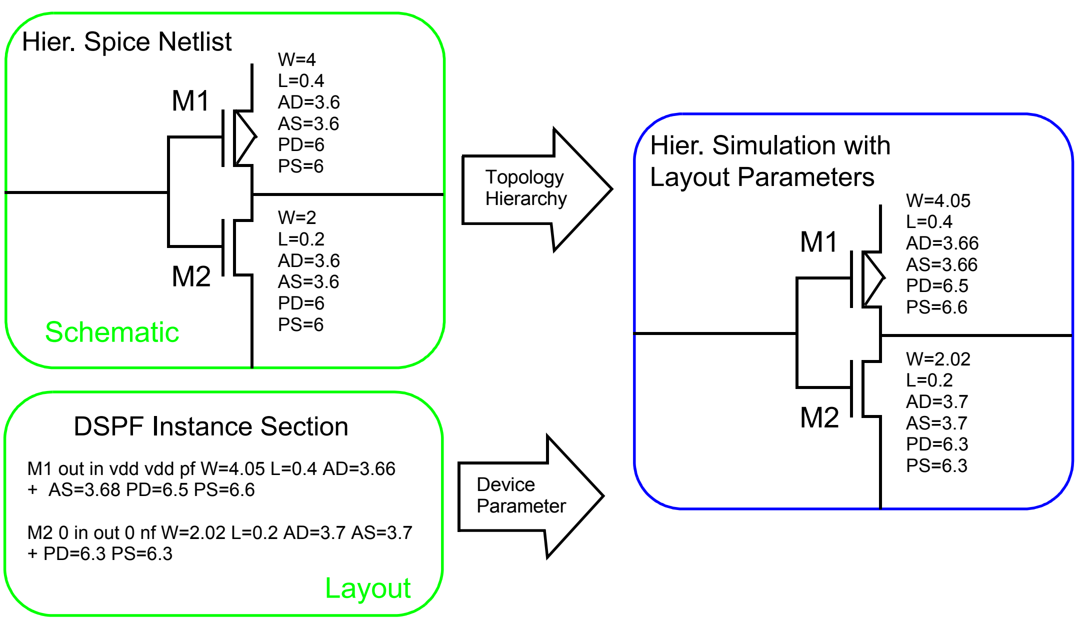
SPICE Syntax

```
.usim_opt dpf=”<instance|subckt> file”
```

Description

The `dpf` option specifies how a parasitic DPF file is loaded into the Virtuoso UltraSim simulator. You can also use `dpf` to stitch the instance section of a DSPF file, causing the parasitic resistors and capacitors to be ignored. The simulator supports backannotation of DPF files. Figure 4-3 on page 258 shows the process of DPF file stitching using the simulator.

Figure 4-3 Stitching a DPF Parasitic File



The Virtuoso UltraSim simulator preserves the hierarchy of the pre-layout netlist file when stitching DPF files (same as SPF/DSPF files).

Arguments

`path`	The full hierarchical path of the instance for which the parasitic file is prepared. Wildcards are supported (for more information about wildcards, see "Wildcard Rules" on page 55). You can also specify the subcircuit name as the path. All instances of the specified subcircuit are stitched. If the path is not specified, and the parasitic file contains a `.subckt` statement, all instances with the same subcircuit name are stitched (or the simulator assumes that the parasitic file is prepared for the entire design, and the quotation marks can be omitted).
`file`	The name of the parasitic file. The Virtuoso UltraSim simulator can read compressed parasitic files in `gz` format (files need to have the `.gz` extension).

For more information about parsing options for DPF files, refer to "Parsing Options for Parasitic Files" on page 263.

Example

Spectre Syntax:

```
usim_opt dpf="top.spf"
```

SPICE Syntax:

```
.usim_opt dpf="top.spf"
```

or

Spectre Syntax:

```
usim_opt dpf="top.spf.gz"
```

SPICE Syntax:

```
.usim_opt dpf="top.spf.gz"
```

tells the Virtuoso UltraSim simulator to only stitch the instance section of the parasitic file `top.spf`.

spf

Spectre Syntax

```
usim_opt spf="<instance|subckt> file"
```

SPICE Syntax

```
.usim_opt spf="<instance|subckt> file"
```

Description

This option is used to specify how a parasitic DSPF file is loaded into the Virtuoso UltraSim simulator.

Arguments

`path`	The full hierarchical path of the instance for which the parasitic file is prepared. Wildcards are supported (for more information about wildcards, see "Wildcard Rules" on page 55). You can also specify the subcircuit name as the path. All instances of the specified subcircuit are stitched. If the path is not specified, and the parasitic file contains a `.subckt` statement, all instances with the same subcircuit name are stitched (or the simulator assumes that the parasitic file is prepared for the entire design, and the quotation marks can be omitted).
`file_name`	The name of the parasitic file. The Virtuoso UltraSim simulator can read compressed parasitic files in `gz` format (files need to have the `.gz` extension).

Example

Spectre Syntax:

```
usim_opt spf=a.dspf
```

SPICE Syntax:

```
.usim_opt spf=a.dspf
```

Consider another example:

```
.subckt sub1 n1 n2
...
.ends sub1
.subckt sub2 m1 m2
...
.ends sub1
```

```
X1 node1 node2 sub1
X2 node3 node4 sub2
X3 node5 node6 sub2
```

If `.usim_opt spf="X2 parasitic.dspf"` is used, the stitching is limited to instance `X2`. If `.usim_opt spf="sub2 parasitic.dspf"` is used, the simulator applies the stitching to all `sub2` instances, which include `X2` and `X3`.

spef

Spectre Syntax

```
usim_opt spef="<instance|subckt> file"
```

SPICE Syntax

```
.usim_opt spef="<instance|subckt> file"
```

Description

This option is used to specify how a parasitic SPEF is loaded into the Virtuoso UltraSim simulator.

Example

Spectre Syntax:

```
usim_opt spef=top.spef
```

SPICE Syntax:

```
.usim_opt spef=top.spef
```

For more information about parsing options for DSPF and SPEF files, refer to "Parsing Options for Parasitic Files" on page 263.

Hierarchical SPEF

Description

The Virtuoso UltraSim simulator also supports stitching of hierarchical SPEF files produced by the Cadence Assura™ physical verification tool. If you specify the SPEF file that corresponds to the highest level of hierarchy of interest, the corresponding SPEF file for the sub-levels is automatically loaded into the simulator via the `*define` statement (per the convention adopted by Virtuoso UltraSim and Assura in which the first string of the `*define` statement indicates the instance name of the sub-block and the second string indicates the name of the SPEF file for the instance).

In general, the Assura RCX tool names the SPEF file and corresponding DPF file as follows: `blockname.spef` and `blockname.dpf` (for example, `vco.spef` and `vco.dpf`). By default, the Virtuoso UltraSim simulator automatically stitches the corresponding DPF file for each stitched SPEF file, provided the DPF files exists in the path. The `usim_opt spfinstancesection=off` option can be used to turn off stitching of the DPF file, useful when you do not want to stitch the corresponding DPF file.

Example

The top level of the SPEF file is named `top.spef` and contains

```
*define X1 INV
*define X2 INV.spef
```

To invoke stitching, use the following statement (Spectre syntax example):

```
usim_opt spef=top.spef
```

The simulator searches for a file named `INV.spef` for the `X1` and `X2` instances. If `INV.spef` is not found, the simulator issues a warning.

Note: You can specify the path in the `*define` statement and the `.spef` suffix can be omitted.

Parsing Options for Parasitic Files

The Virtuoso UltraSim simulator provides you with the options needed to parse parasitic files. Unless stated otherwise, parsing options are global and are only applicable to parasitic files during stitching (not applicable to pre-layout netlist files).

The simulator supports the following parsing options:

cmin

Spectre Syntax

```
usim_opt cmin=value
```

SPICE Syntax

```
.usim_opt cmin=value
```

Description

A parasitic capacitor less than `cmin` is discarded (open circuited) during RC stitching. The default is 0.

Note: Capacitors in the pre-layout are not affected because they are not stitched.

Example

Spectre Syntax:

```
usim_opt cmin=1ff
```

SPICE Syntax:

```
.usim_opt cmin=1ff
```

tells the Virtuoso UltraSim simulator to discard any parasitic capacitors < 1 ff during stitching.

cmingnd

Spectre Syntax

```
usim_opt cmingnd=value
```

SPICE Syntax

```
.usim_opt cmingnd=value
```

Description

A parasitic coupling capacitor less than `cmingnd` is grounded during RC stitching (default is 0).

Example

Spectre Syntax:

```
usim_opt cmingnd=1ff
```

SPICE Syntax:

```
.usim_opt cmingnd=1ff
```

tells the Virtuoso UltraSim simulator to ground a parasitic coupling capacitor during stitching if the capacitor is < 1 ff.

cmingndratio

Spectre Syntax

```
usim_opt cmingndratio=value
```

SPICE Syntax

```
.usim_opt cmingndratio=value
```

Description

A parasitic coupling capacitor is grounded during RC stitching if the ratio of its value to the total node capacitance on both sides is less than `cmingndratio` (default is 0).

Note: The pre-layout capacitors are not affected.

Example

Spectre Syntax:

```
usim_opt cmingndratio=0.1
```

SPICE Syntax:

```
.usim_opt cmingndratio=0.1
```

tells the Virtuoso UltraSim simulator to ground a parasitic coupling capacitor during stitching if the capacitor has a ratio < 0.1, when compared to the total node capacitance on both sides of the capacitor.

dpfautoscale

Spectre Syntax

```
usim_opt dpfautoscale=on|off
```

SPICE Syntax

```
.usim_opt dpfautoscale=on|off
```

Description

This option controls the behavior of the dpfscale option. When `dpfautoscale` is set to `on`, `dpfscale` is set to the same value as the parameter `scale`. When `dpfautoscale` is set to `off`, dpfscale is independent of the parameter `scale`. The `dpfautoscale` option is `off` by default.

Example

Spectre Syntax:

```
usim_opt dpfautoscale=on
```

SPICE Syntax:

```
.usim_opt dpfautoscale=on
```

tells the Virtuoso UltraSim simulator to use the value of the parameter `scale` for `dpfscale`.

dpfscale

Spectre Syntax

```
usim_opt dpfscale=value
```

SPICE Syntax

```
.usim_opt dpfscale=value
```

Description

This option specifies the scale factor for device geometry parameters (default is 1.0).

Note: The pre-layout device parameters are not affected by `dpfscale`.

Example

Spectre Syntax:

```
usim_opt dpfscale=2
```

SPICE Syntax:

```
.usim_opt dpfscale=2
```

tells the Virtuoso UltraSim simulator to scale all the device geometry parameters by a multiple of two during stitching.

rmax

Spectre Syntax

```
usim_opt rmax=value
```

SPICE Syntax

```
.usim_opt rmax=value
```

Description

All parasitic resistors with resistances larger than the value specified by this option are treated as open-circuited during stitching. This is useful to cut parasitic resistors that are unreasonably large. This does not affect any resistors in pre-layout netlist. The default value of the option is `infinity`.

This option is usually used when the resistance values appear suspicious, that is, non-physical. In general, this indicates that the extraction is questionable. Large resistors, such as 1.0e+9, may cause simulation problems. The option `rmax` helps bypass any extraction problem and continue with simulation.

Example

Spectre Syntax:

```
usim_opt rmax=1.0e+9
```

SPICE Syntax:

```
.usim_opt rmax=1.0e+9
```

tells the Virtuoso UltraSim simulator to treat all resistors with resistances greater than 1.0e+9 as open-circuited and cut such resistors.

rmaxlayer

Spectre Syntax

```
usim_opt rmaxlayer=value&layername
```

SPICE Syntax

```
.usim_opt rmaxlayer=value&layername
```

Description

This option applies the `rmax` value on the parasitic resistors of the specified layer. Multiple statements can be specified. The option is used when the extracted resistance values are unreasonably large on certain layers.

Examples

Spectre Syntax

```
usim_opt rmaxlayer=1000000&ptab
usim_opt rmaxlayer=1000000&ptab0
usim_opt rmaxlayer=1000000&ntap
```

SPICE Syntax

```
.usim_opt rmaxlayer=1000000&ptab
.usim_opt rmaxlayer=1000000&ptab0
.usim_opt rmaxlayer=1000000&ntap
```

tells the Virtuoso UltraSim simulator to treat any parasitic resistors on layers `ptab`, `ptab0`, and `ntap` with value greater than 1000000 Ohm as open circuit for layers.

rmin

Spectre Syntax

```
usim_opt rmin=value
```

SPICE Syntax

```
.usim_opt rmin=value
```

Description

If `rmin` is specified, a parasitic resistor less than `rmin` is shorted during RC stitching (default is 0).

Note: The pre-layout resistors are not affected by `rmin`.

Example

Spectre Syntax:

```
usim_opt rmin=1
```

SPICE Syntax:

```
.usim_opt rmin=1
```

tells the Virtuoso UltraSim simulator to short circuit parasitic resistors with a value < 1 ohm during RC stitching.

rminlayer

Spectre Syntax

```
usim_opt rminlayer=value&layername
```

SPICE syntax

```
.usim_opt rminlayer=value&layername
```

Description

This option applies the `rmin` value on parasitic resistors of the specified layer. Multiple statements can be specified. The option is used when the extracted resistance values are unreasonably small on certain layers.

Examples

Spectre syntax:

```
usim_opt rminlayer=0.1001&nwell
usim_opt rminlayer=0.1001&dnwell
usim_opt rminlayer=0.1001&sub
```

SPICE syntax

```
.usim_opt rminlayer=0.1001&nwell
.usim_opt rminlayer=0.1001&dnwell
.usim_opt rminlayer=0.1001&sub
```

tells the Virtuoso UltraSim simulator to short any parasitic resistors on layers `nwell`, `dnwell` and `sub` with value less than 0.1001 Ohm.

rvmin

Spectre Syntax

```
usim_opt rvmin=value
```

SPICE Syntax

```
.usim_opt rvmin=value
```

Description

If `rvmin` is specified, a power net parasitic resistor less than `rvmin` is short circuited during RC stitching (default is 0).

Note: The pre-layout rail resistors are not affected by `rvmin`.

Example

Spectre Syntax:

```
usim_opt rvmain=1
```

SPICE Syntax:

```
.usim_opt rvmain=1
```

tells the Virtuoso UltraSim simulator to short circuit the parasitic rail resistors with a value < 1 ohm during RC stitching.

speftriplet

Spectre Syntax

```
usim_opt speftriplet=1|2|3
```

SPICE Syntax

```
.usim_opt speftriplet=1|2|3
```

Description

This option specifies which value should be used for stitching in the SPEF file. This is effective only when the values in the SPEF file are represented by triplets (for instance, 0.325:0.41:0.495). The default is `2`.

Example

Spectre Syntax:

```
usim_opt speftriplet=1
```

SPICE Syntax:

```
.usim_opt speftriplet=1
```

tells the Virtuoso UltraSim simulator to choose the first of the triplet values in the SPEF file for stitching.

spfbusdelim

Spectre Syntax

```
usim_opt spfbusdelim="\""
```

SPICE Syntax

```
.usim_opt spfbusdelim="\""
```

Description

This option specifies the bus delimiter in parasitic files, and is applicable to SPEF, DSPF, and cap files. It also affiliates bus delimiter matching between the pre-layout netlist and the SPEF/DSPF files, and is used when the bus delimiter in the pre-layout netlist file is different from what is specified in the parasitic file.

The Virtuoso UltraSim simulator replaces the bus delimiter in the parasitic file with information from `spfbusdelim`. Normally `spfbusdelim` is set the same as the pre-layout netlist file (default is `<>`).

Notes:

- For SPEF/DSPF files, when the `bus_delimiter` statement is not used in the parasitic files, the simulator automatically converts braces `{}` and square brackets `[]` into angle brackets `<>`.
- To define the delimiter as ″ or \, the forward slash (\) symbol is required (for example, `usim_opt spfbusdelim=″\″″`).

Examples

1. The name in the pre-layout netlist file is `qn<5>` and the name in the SPEF file is `qn[5]`. If there is no `bus_delimiter` statement, `qn[5]` is automatically converted to `qn<5>` since qn<5> exists in the pre-layout netlist file. The simulator finds the match and stitching is successful.

   If `bus_delimiter : []` is used in the SPEF file, `qn[5]` does not change because the Virtuoso UltraSim simulator uses the definition of `bus_delimiter` specified in the SPEF file. However, there is a mismatch in the pre-layout netlist file because the name is `qn<5>`. You can use the `usim_opt spfbusdelim=<>` option to resolve this problem.

2. The name in the pre-layout netlist file is `qn[5]` and the name in the SPEF file is `qn{5}`. If the `bus_delimiter : {}` statement is located in the SPEF file, `qn{5}` remains the

same (that is, the name is not converted to `qn<5>`). There is an obvious mismatch between the pre-layout netlist file and the SPEF file. You can set `usim_opt spfbusdelim=[]` to resolve this problem.

spfcaponly

Spectre Syntax

```
usim_opt spfcaponly=no|yes
```

SPICE Syntax

```
.usim_opt spfcaponly=no|yes
```

Description

This option controls whether or not to treat a DSPF/SPEF file as a capfile.

Arguments

`yes`	The Virtuoso UltraSim simulator stitches the parasitic file as a node capacitance file (that is, the simulator only parses and backannotates the total node capacitance).
`no`	The simulator performs a routine parse and backannotation of the DSPF/SPEF file (default).

Example

Spectre Syntax:

```
usim_opt spfcaponly=yes
```

SPICE Syntax:

```
.usim_opt spfcaponly=yes
```

tells the simulator to read in and backannotate only the total node capacitance of each net.

spfcrossccap

Spectre Syntax

```
usim_opt spfcrossccap=on|off|search|onsearch
```

SPICE Syntax

```
.usim_opt spfcrossccap=on|off|search|onsearch
```

Description

The `spfcrossccap` option specifies stitching of matched and unmatched coupling capacitors. In general, a coupling capacitor is instantiated in both nets of the capacitor and is called a matched coupling capacitor (otherwise, it is called an unmatched coupling capacitor).

Arguments

`on`	The Virtuoso UltraSim simulator stitches matched cross coupling capacitors and ignores unmatched coupling capacitors. **Note:** If there are no unmatched cross coupling capacitors, `on` is equivalent to `onsearch` and `off` is equivalent to `search`.
`off`	The simulator grounds cross coupling capacitors (default).
`search`	The simulator grounds matched and unmatched coupling capacitors.
`onsearch`	The simulator stitches matched and unmatched coupling capacitors.

Example

Spectre Syntax:

```
usim_opt spfcrossccap=onsearch
```

SPICE Syntax:

```
.usim_opt spfcrossccap=onsearch
```

tells the Virtuoso UltraSim simulator to find matched and unmatched coupling capacitors and to stitch them as coupling capacitors.

spffingerdelim

Spectre Syntax

```
usim_opt spffingerdelim="$"
```

SPICE Syntax

```
.usim_opt spffingerdelim="$"
```

Description

The `spffingerdelim` option specifies the fingered delimiter symbol (default is `null`). The original device can be split into several devices. For example, a large MOSFET device can be split into several smaller MOSFETs that are connected in parallel. The names for these devices are created by adding the finger postfix to the name of the original device. This postfix normally consists of integers and should be separated from the original name by a special symbol (finger delimiter). If the RC extractor uses any symbol as a finger delimiter, it needs to be specified by `spffingerdelim`.

Note: To define the delimiter as `"` or `\`, the forward slash (`\`) symbol is required (for example, `usim_opt spffingerdelim="\""`).

Example

Spectre Syntax:

```
usim_opt spffingerdelim=@
```

SPICE Syntax:

```
.usim_opt spffingerdelim=@
```

tells the Virtuoso UltraSim simulator that the DSPF file uses the `@` symbol as the finger delimiter.

Stitching of Parameterized Subcircuit Instances

Parameterized subcircuits with one level of hierarchy are often used in device modeling of advanced technologies (for example, inline subcircuits in Spectre MOSFET models for 65 nm and below). The Virtuoso UltraSim simulator can also be used to stitch this type of instance.

For example,

In the pre-layout netlist file, `xM1` is an instance of the parameterized subcircuit `nmos`.

```
xM1 drn gate src gnd nmos W=1u L=0.5u m=1
.subckt nmos drn gate src bulk W=2u L=1u m=1
.param W0=W*2 L0=L+1 Main drn g1 src bulk nfet w=W0 L=L0
R1 gate g1 2
.model nmos nfet level=49 version=3.2 ...
.ends
```

The instance section of the DSPF file contains the following:

```
*Instance Section

xM1 xM1:drn xM1:gate xM1:src xM1:bulk NMOS w=1.4u L=0.6u m=1
```

After stitching, the `w` (width) and `l` (length) parameters for `xM1.Main` are 2.8 u and 1.6 u, respectively.

spfhierdelim

Spectre Syntax

```
usim_opt spfhierdelim="."
```

SPICE Syntax

```
.usim_opt spfhierdelim="."
```

Description

This option specifies the hierarchical delimiter in the DSPF and SPEF files, and the cap file. If `spfhierdelim` and the hierarchical `divider` statement in the DSPF file are set, the hierarchical `divider` in the DSPF and SPEF file has a higher priority (the default is `/`).

Note: To define the delimiter as `"` or `\`, the forward slash (`\`) symbol is required (for example, `usim_opt spfhierdelim="\""`).

Example

Spectre Syntax:

```
usim_opt spfhierdelim=.
```

SPICE Syntax:

```
.usim_opt spfhierdelim=.
```

tells the Virtuoso UltraSim simulator to treat the period (`.`) symbol in the cap file as the hierarchical delimiter for cap file stitching.

spfinstancesection

Spectre Syntax

```
usim_opt spfinstancesection=on|off
```

SPICE Syntax

```
.usim_opt spfinstancesection=on|off
```

Description

This option controls the backannotation of device parameters in the instance section of the DSPF file. It is only applicable to the DSPF file specified by the `spf` option, and it does not apply to the SPEF, DPF, or cap files. If `spfinstancesection` is turned off, the instance section is ignored (that is, the device parameters are not changed during stitching). The default is `on`.

Examples

In the following Spectre syntax example

```
usim_opt spf=a.dspf spfinstancesection=off dpf=a.dspf
```

tells the Virtuoso UltraSim simulator not to stitch the instance section for `a.dspf` because the `a.dspf` file is first defined by the `spf` option and `spfinstancesection` is set to `off`. The simulator then stitches the device section because `a.dspf` is defined next by the `dpf` option.

The following SPICE syntax example has the same outcome as the previous example.

```
.usim_opt spf=a.dspf (with default pfinstancesection being on)
```

spfkeepbackslash

Spectre Syntax

```
usim_opt spfkeepbackslash=on|off
```

SPICE Syntax

```
.usim_opt spfkeepbackslash=on|off
```

Description

The `spfkeepbackslash` option defines the back slash (`\`) symbol, used for net, instance, and other design component names, as a normal character instead of an *Escape* character (default is `off`).

Example

Spectre Syntax:

```
usim_opt spfkeepbackslash=on
```

SPICE Syntax

```
.usim_opt spfkeepbackslash=on
```

tells the Virtuoso UltraSim simulator to treat the `\` symbol as a normal character.

spfnamelookup

Spectre Syntax

```
usim_opt spfnamelookup=matchx
```

SPICE Syntax

```
.usim_opt spfnamelookup=matchx
```

Description

This option specifies the Virtuoso UltraSim simulator name lookup method (default is `none`). If this option is set, the simulator tries to find matching elements in the following order, until a match is found:

1. "As is" match
2. Match with an extra leading `X` character in every level of the name hierarchy
3. Match without a leading `X` character in every level of the name hierarchy

Note: `spfnamelookup` also takes care of case sensitivity.

A valuable application of `spfnamelookup` is to stitch a HSPICE DSPF file with a Spectre netlist or Spectre-like SPEF file with a HSPICE netlist file. There is a different convention regarding subcircuit names in HSPICE and Spectre: HSPICE requires a leading `X` in the subcircuit instance name, whereas Spectre does not have this restriction. The `spfnamelookup` option can also help find matches when HSPICE and Spectre netlist files are mixed.

It is important to note that using this option slows down the stitching process. Also, the simulator does not change the instance name in the parasitic file when running the `spfnamelookup` option.

Example

Spectre Syntax:

```
usim_opt spfnamelookup=matchx
```

SPICE Syntax:

```
.usim_opt spfnamelookup=matchx
```

tells the Virtuoso UltraSim simulator to find matching elements, in the order specified above (if a match is found, the simulator continues to stitch the parasitics, otherwise an error message is issued).

spfrcnet

Spectre Syntax

```
usim_opt spfrcnet=net_name
```

SPICE Syntax

```
.usim_opt spfrcnet=net_name
```

Description

This option specifies the name of the net to be stitched with parasitic resistors and capacitors. The other nets are stitched with lumped capacitances. Multiple nets can be specified on separate lines.

Example

Spectre Syntax:

```
usim_opt spfrcnet=vcc
usim_opt spfrcnet=gnd
```

SPICE Syntax:

```
.usim_opt spfrcnet=vcc
.usim_opt spfrcnet=gnd
```

tells the Virtuoso UltraSim simulator to stitch parasitic resistors and capacitors for both *vcc* and *gnd* nets and to stitch all other nets with lumped capacitances.

spfrcreduction

Spectre Syntax

```
usim_opt spfrcreduction=on|off
```

SPICE Syntax

```
.usim_opt spfrcreduction=on|off
```

Description

The `spfrcreduction` option controls RC reduction during RC stitching. If `spfrcreduction` is `on` (default), the RC reduction algorithm can be applied during stitching (depending on how the RC reduction options are set). Using this option can improve simulation performance and memory. If `spfrcreduction` is `off`, RC reduction is not applied during stitching, regardless of the RC reduction option settings.

Example

Spectre Syntax:

```
usim_opt spfrcreduction=on
```

SPICE Syntax:

```
.usim_opt spfrcreduction=on
```

tells the Virtuoso UltraSim simulator to perform RC reduction during RC stitching if the RC reduction options, such as `postl (1|2|3|4)`, are enabled. If `postl=0`, the simulator does not perform RC reduction, even it `spfrcreduction=on`.

spfrecover

Spectre Syntax

```
usim_opt spfrecover=0|1|2|3|4
```

SPICE Syntax

```
.usim_opt spfrecover=0|1|2|3|4
```

Description

This option controls how the Virtuoso UltraSim simulator handles erroneous parasitic nets. If the definition of a net in a parasitic file contains an error, you can use the `spfrecover` option to control how this erroneous net is stitched.

Arguments

`0` Erroneous nets are not stitched (default).

`1` Only the total capacitance is stitched for an erroneous net.

`2` The Virtuoso UltraSim simulator takes some corrective measures for erroneous nets. For an instance:

- Specified in the parasitic files, but not found in the pre-layout netlist file, the simulator tries to accept nodes associated with this instance
- Located in the pre-layout netlist file, but missing in the parasitic file, the simulator maintains the original connectivity

After the simulator makes these corrections, if the net still has errors in its parasitic definitions, only the total capacitance of the net is stitched.

`3` The simulator tries to stitch swapped devices and devices with gate swapping, along with the elements described in level `2`.

`4` To speed up the stitching process, the simulator tries to stitch what can be stitched and discard any devices and subnodes that have errors without any repair. This is recommended for big nets, such as for the power nets of a full chip design when the performance of `spfrecover=2|3` is not good enough.

Example

Spectre Syntax:

```
usim_opt spfrecover=3
```

SPICE Syntax:

```
.usim_opt spfrecover=3
```

tells the Virtuoso UltraSim simulator to accept nodes associated with instances not found in the pre-layout netlist file. The simulator keeps the original connectivities of all instances that are missing in the DSPF/SPEF file, but exist in the pre-layout netlist file, and also tries to stitch swapped devices. If erroneous nets still exist after all of the corrective measures are taken by the simulator, it then tries to stitch the total capacitances.

spfscalec

Spectre Syntax

```
usim_opt spfscalec=value
```

SPICE Syntax

```
.usim_opt spfscalec=value
```

Description

This option specifies the scale factor for parasitic capacitors (default is 1.0).

Note: The pre-layout capacitors are not affected by `spfscalec`.

Example

Spectre Syntax:

```
usim_opt spfscalec=1.5
```

SPICE Syntax:

```
.usim_opt spfscalec=1.5
```

tells the Virtuoso UltraSim simulator to scale all the capacitors in the parasitic file by 1.5 times during stitching.

spfscaler

Spectre Syntax

```
usim_opt spfscaler=value
```

SPICE Syntax

```
.usim_opt spfscaler=value
```

Description

This option specifies the scale factor for parasitic resistors (default is 1.0).

Note: The pre-layout resistors are not affected by `spfscaler`.

Example

Spectre Syntax:

```
usim_opt spf spfscaler=2.0
```

SPICE Syntax:

```
.usim_opt spf spfscaler=2.0
```

tells the Virtuoso UltraSim simulator to scale all the resistors in the parasitic file by a multiple of two during stitching.

spfserres

Spectre Syntax

```
usim_opt spfserres=instance_name
```

SPICE Syntax

```
.usim_opt spfserres=instance_name
```

Description

This option specifies the instance name of the resistor that has serial fingers.

Use this option to stitch serial resistance fingers correctly. At times, a schematic resistor with large resistance value is represented as serpentine resistors in the layout, as shown below:

These serpentine resistors are extracted as serial resistor fingers for the schematic resistor.

Note: You can specify multiple instance names using multiple `spfserres` options.

Example

Spectre Syntax:

```
usim_opt spfserres=X1.XRR1
```

In the above example, `X1.XRR1` is the instance name of a schematic resistor, which is represented by resistors that have two or more serial fingers. This setting ensures the correct stitching of the `X1.XRR1` schematic resistor.

SPICE Syntax:

```
.usim_opt spfserres=X1.XRR1
```

In the above example, `X1.XRR1` is the instance name of a schematic resistor, which is represented by resistors that have two or more serial fingers. This setting ensures the correct stitching of the `X1.XRR1` schematic resistor.

spfserresmod

Spectre Syntax

```
usim_opt spfserresmod=model_name|subckt_name
```

SPICE Syntax

```
.usim_opt spfserresmod=model_name|subckt_name
```

Description

This option specifies the model or subckt name for resistors that have serial fingers. When a model or a subckt is used for a resistor, it typically signifies the boundary of extraction.

Use this option to stitch serial resistance fingers correctly. At times, a schematic resistor with large resistance value is represented as serpentine resistors in the layout, as shown below:

These serpentine resistors are extracted as serial resistor fingers for the schematic resistor.

Note: You can specify multiple model names and/or subckt names using multiple `spfserresmod` options.

Example

Spectre Syntax:

```
usim_opt spfserresmod=POLY
```

In the above example, `POLY` is the model name. This setting ensures the correct stitching of a schematic resistor, which is represented by resistors that have two or more serial fingers, and is an instance of the model `POLY`.

SPICE Syntax:

```
.usim_opt spfserresmod=POLY
```

In the above example, `POLY` is the model name. This setting ensures the correct stitching of a schematic resistor, which is represented by resistors that have two or more serial fingers, and is an instance of the model `POLY`.

spfsplitfinger

Spectre Syntax

```
usim_opt spfsplitfinger=on|off
```

SPICE Syntax

```
.usim_opt spfsplitfinger=on|off
```

Description

This option specifies whether all fingers are simulated as individual devices (`spfsplitfinger=on`) or all fingers are merged into one device and average values, such as width and length, are applied to the device (`spfsplitfinger=off`).

spfswapterm

Spectre Syntax

```
usim_opt spfswapterm="terminal1 terminal2 subckt"
```

SPICE Syntax

```
.usim_opt spfswapterm="terminal1 terminal2 subckt"
```

Description

This option allows the swapping of *terminal1* and *terminal2* of *subckt* during stitching. UltraSim stitching considers terminals swappable for the following primitive devices:

- Drain and source of a primitive mosfet
- Two terminals of a capacitor
- Two terminals of a resistor

This sometimes causes stitching problems if the primitive device, say a mosfet, is modeled by a parameterized subckt that is complicated. For example,

```
.subckt mosfet d g s b w1=1 l1=1
m1 d g s b nmos w=w1 l=l1
d1 s b didoe
d2 d b didoe
.ends
```

In this example, due to the existence of diodes, stitching considers `d` and `s` to be non-swappable. However, using the option `spfswapterm`, you can define `d` and `s` to be swappable for the purpose of stitching.

Example

Spectre Syntax:

```
usim_opt spfswapterm="n1 n2 npres"
```

SPICE Syntax:

```
.usim_opt spfswapterm="n1 n2 npres"
```

tells the Virtuoso UltraSim simulator that the terminals *n1* and *n2* of the subckt *npres* are swappable terminals.

spfxtorintop

Spectre Syntax

```
usim_opt spfxtorintop=yes|no
```

SPICE Syntax

```
.usim_opt spfxtorintop=yes|no
```

Description

This option helps stitching of primitive elements at the top-level within the extraction scope. It is used along with spfxtorprefix. When `spfxtorintop` is set to `yes`, the substitution specified by `spfxtorprefix` is applicable not only to top-level primitive elements but also to top-level instances names.

Example

This is an example of primitive elements at the top-level within the extraction scope. Consider that subckt `s1` is extracted. This subckt is defined in the pre-layout as:

```
subckt s1 in out vss vdd
N1 out1 in vss vss nmos w=1 l=0.5
P1 out1 in vdd vdd pmos w=2 l=0.5
X1 out1 out  vss vdd inv
ends



subckt in out inv
M1 out in vss vss nmos w=2 l=0.5
M2 out in vdd vdd pmos w=4 l=0.5
ends
```

In the extracted DSPF for subckt `s1`, the primitive element `N1` is at the top-level of the extraction scope subckt `s1`. If the element name in the DSPF file for element `N1` is `MN1`, the following option helps the stitching process find the right match:

```
.usim_opt spfxtorintop=yes
.usim_opt spfxtorprefix="MN N"
```

spfxtorprefix

Spectre Syntax

```
usim_opt spfxtorprefix="<substring> [<replace_substring>]"
```

SPICE Syntax

```
.usim_opt spfxtorprefix="<substring> [<replace_substring>]"
```

Description

The `spfxtorprefix` option replaces `substring` (that is, the `xtorprefix` used in the RC extractor) with `replace_substring`. Based on the requirement that a DSPF file must be used as a HSPICE netlist file, all MOSFETs need to begin with the `m` symbol, all diodes with the `d` symbol, and so on. The hierarchical names for devices must contain leading symbols based on type, in order for HSPICE to recognize the names. Some RC extractors append a different prefix to the hierarchical name, such as `MX` (this prefix is considered a `xtorprefix`).

To match the device names in the parasitic file with the names in the pre-layout netlist file, you need to specify `spfxtorprefix` to be the same as what is used by the RC extractor. If it becomes necessary to change the `xtorprefix` substring to a different substring, specify the `replace_substring` option. There is no limit to the number of `xtorprefices` that you can specify (default is `none`).

Example

Spectre Syntax:

```
usim_opt spfxtorprefix="MX_ X" spfxtorprefix="D" spfxtorprefix="R XI"
```

SPICE Syntax:

```
.usim_opt spfxtorprefix="MX_ X" spfxtorprefix="D" spfxtorprefix="R XI"
```

tells the Virtuoso UltraSim simulator to replace all instance names starting with `MX_` with `X`, to remove the `D` prefix from all instance names starting with `D`, and to replace prefix `R` in all instance names with `XI`.

Selective RC Backannotation

The Virtuoso UltraSim simulator supports a set of options for selective RC backannotation. All the options listed in this section are global.

The simulator supports the following selective RC backannotation options:

- spfactivenet on page 302
- spfactivenetfile on page 303
- spfchlevel on page 304
- spfcnet on page 305
- spfcnetfile on page 306
- spfhlevel on page 307
- spfnetcmin on page 308
- spfrcnetfile on page 309
- spfskipnet on page 310
- spfskipnetfile on page 311
- spfskippwnet on page 312
- spfskipsignet on page 313

spfactivenet

Spectre Syntax

```
usim_opt spfactivenet=net_name
```

SPICE Syntax

```
.usim_opt spfactivenet=net_name
```

Description

This option specifies the nets to be stitched by name (default is `none`). All other nets are ignored and will not be stitched. Wildcards are supported and you can specify as many nets as needed.

For more information about wildcards, see “Wildcard Rules” on page 55.

Note: When multiple nets are specified, the Virtuoso UltraSim simulator requires the nets to be on separate lines (see example below).

Example

Spectre Syntax:

```
usim_opt spfactivenet=nodeA
usim_opt spfactivenet=nodeB
```

SPICE Syntax:

```
.usim_opt spfactivenet=nodeA
.usim_opt spfactivenet=nodeB
```

tells the Virtuoso UltraSim simulator to stitch the parasitics associated only with `nodeA` and `nodeB` (all other nets are not stitched).

spfactivenetfile

Spectre Syntax

```
usim_opt spfactivenetfile=<file_name>
```

SPICE Syntax

```
.usim_opt spfactivenetfile=<file_name>
```

Description

This option allows you to specify the nets that need to be stitched as a list in a text file called `file_name` (only one file can be specified).

It is important to note if `spfactivenet` or `spfactivenetfile` is defined, only the specified nets are stitched. Also, you can use the `acheck` statement to generate the active nets file.

Example

Spectre Syntax:

```
usim_opt spfactivenetfile=nets.tex
```

SPICE Syntax:

```
.usim_opt spfactivenetfile=nets.tex
```

`nets.tex` file format:

```
netA
netB
netC
```

tells the Virtuoso UltraSim simulator to stitch the parasitics associated with the nets specified in the `nets.tex` file, and to ignore the other nets.

spfchlevel

Spectre Syntax

```
usim_opt spfchlevel=value
```

SPICE Syntax

```
.usim_opt spfchlevel=value
```

Description

The `spfchlevel` option allows you to select the net for stitching by its hierarchy level. If the net name has a hierarchy level less than `spfchlevel`, the net is stitched (otherwise, only the total capacitance is added to the net node). The default is 1000 or the lowest level.

Example

Spectre Syntax:

```
usim_opt spfchlevel=10
```

SPICE Syntax:

```
.usim_opt spfchlevel=10
```

tells the Virtuoso UltraSim simulator to stitch nets that have a hierarchical level < 10.

spfcnet

Spectre Syntax

```
usim_opt spfcnet=net_name
```

SPICE Syntax

```
.usim_opt spfcnet=net_name
```

Description

This option specifies the net that will have its total capacitance stitched and all other parasitic components associated with this net are ignored (default is `none`). Wildcards are supported and you can specify as many nets as needed.

For more information about wildcards, see "Wildcard Rules" on page 55.

Example

Spectre Syntax:

```
usim_opt spfcnet=netA
```

SPICE Syntax:

```
.usim_opt spfcnet=netA
```

tells the Virtuoso UltraSim simulator to stitch only the total capacitance of `netA` and to ignore other parasitics associated with `netA`.

spfcnetfile

Spectre Syntax

```
usim_opt spfcnetfile=<file_name>
```

SPICE Syntax

```
.usim_opt spfcnetfile=<file_name>
```

Description

This option has the same functionality as spfcnet with the exception that all the nets are specified as a list in a file named `file_name`. Only one file can be specified.

Example

Spectre Syntax:

```
usim_opt spfcnetfile=nets.tex
```

SPICE Syntax:

```
.usim_opt spfcnetfile=nets.tex
```

`nets.tex` file format:

```
netA
netB
netC
```

tells the Virtuoso UltraSim simulator to stitch only the total node capacitance for all nets specified in the `nets.tex` file, and to ignore all the other parasitics associated with these nets.

spfhlevel

Spectre Syntax

```
usim_opt spfhlevel=value
```

SPICE Syntax

```
.usim_opt spfhlevel=value
```

Description

This option allows you to select the net for stitching by its hierarchy level. If a net name has a hierarchy level more than or equal to `spfhlevel`, all the parasitics associated with the net are stitched (otherwise, only the total capacitance is added to the net node). The default is -1 or the top level.

Example

Spectre Syntax:

```
usim_opt spfhlevel=10
```

SPICE Syntax:

```
.usim_opt spfhlevel=10
```

spfnetcmin

Spectre Syntax

```
usim_opt spfnetcmin=value
```

SPICE Syntax

```
.usim_opt spfnetcmin=value
```

Description

The `spfnetcmin` option allows you to select the net for stitching by the value of its total node capacitance. If the total node capacitance exceeds `spfnetcmin`, the net is stitched. That is, all the parasitics associated with the net are stitched correctly (otherwise, only the total capacitance is added to the net node). The default is 0.0.

Example

Spectre Syntax:

```
usim_opt spfnetcmin=1ff
```

SPICE Syntax:

```
.usim_opt spfnetcmin=1ff
```

tells the Virtuoso UltraSim simulator to stitch the nets, including all the parasitic components of the nets, with a total node capacitance > 1 ff (if the total capacitance is less than or equal to 1 ff, only the total capacitance is stitched).

spfrcnetfile

Spectre Syntax

```
usim_opt spfrcnetfile=<file_name>
```

SPICE Syntax

```
.usim_opt spfrcnetfile=<file_name>
```

Description

This option allows you to specify the nets that need to be stitched as a list in the specified file_name (only one file can be specified).

Important

If `spfrcnet` or `spfrcnetfile` is defined, only the specified nets are stitched with RC and the remaining nets in the DSPF/SPEF files are stitched with C-only.

Example

Spectre Syntax:

```
usim_opt spfrcnetfile=nets.tex
```

SPICE Syntax:

```
.usim_opt spfrcnetfile=nets.tex
nets.tex file takes the following format:
netA
netB
netC
```

tells the Virtuoso UltraSim simulator to stitch the parasitic RC associated with the nets specified in the `nets.text` file, and to stitch C only for the rest of the nets in the DSPF file.

spfskipnet

Spectre Syntax

```
usim_opt spfskipnet=net_name
```

SPICE Syntax

```
.usim_opt spfskipnet=net_name
```

Description

This option specifies the net to be skipped by name and all parasitic components of the net are not stitched (default is `none`). Wildcards are supported and you can specify as many nets as needed.

For more information about wildcards, see "Wildcard Rules" on page 55.

Note: If multiple nets need to be specified, the nets must be placed in separate lines.

Example

Spectre Syntax:

```
usim_opt spfskipnet=nodeA
```

SPICE Syntax:

```
.usim_opt spfskipnet=nodeA
```

tells the Virtuoso UltraSim simulator not to stitch the parasitics associated with `nodeA`.

spfskipnetfile

Spectre Syntax

```
usim_opt spfskipnetfile=file_name
```

SPICE Syntax

```
.usim_opt spfskipnetfile=file_name
```

Description

This option allows you to specify the nets to be skipped as a list in a text file called `file_name`. Only one file can be specified.

Example

Spectre Syntax:

```
usim_opt spfskipnetfile=nets.tex
```

SPICE Syntax:

```
.usim_opt spfskipnetfile=nets.tex
```

`nets.tex` file format:

```
netA
netB
netC
```

tells the Virtuoso UltraSim simulator not to stitch the parasitics associated with `netA`, `netB`, and `netC`.

Note: The net names in the file need to be located on separate lines.

spfskippwnet

Spectre Syntax

```
usim_opt spfskippwnet=on|off
```

SPICE Syntax

```
.usim_opt spfskippwnet=on|off
```

Description

This option allows you to skip stitching for parasitics associated with the nets that are connected to DC voltage sources. These nets usually contain a large number of parasitics, and in most types of analyses, do not affect the simulation results and can be omitted. If the stitching of power nets is important, set `spfskippwnet` to `off`. The default is `on`.

Example

Spectre Syntax:

```
usim_opt spfskippwnet=off
```

SPICE Syntax:

```
.usim_opt spfskippwnet=off
```

tells the Virtuoso UltraSim simulator to stitch the parasitics of nets connected to DC voltages.

spfskipsignet

Spectre Syntax

```
usim_opt spfskipsignet=on|off
```

SPICE Syntax

```
.usim_opt spfskipsignet=on|off
```

Description

The `spfskipsignet` option allows you to skip stitching for parasitics associated with signal nets (default is `off`).

Note: If `spfskippwnet=on` and `spfskipsignet=on`, no single nets are stitched.

Example

Spectre Syntax:

```
usim_opt spfskipsignet=on
```

SPICE Syntax:

```
.usim_opt spfskipsignet=on
```

tells the Virtuoso UltraSim simulator to ignore stitching parasitics for all the signal nets.

Error/Warning Message Control Options for Stitching

The Virtuoso UltraSim simulator error and warning messages generated during stitching are printed in the `netlistname.spfrpt` file, and the cumulative (statistics) report is printed to the screen and a log file. The default number of error and warning messages is limited to 50. If you need to see more messages, increase the limit by using the spfmaxerrormsg or spfmaxwarnmsg options.

Errors that cannot be corrected are issued as error messages and correctable errors are issued as warning messages. The Virtuoso UltraSim simulator is able to print out notes that you can use to track the stitching process. You can also control the level of reporting with the spferrorreport option.

The Virtuoso UltraSim simulator supports the following error and warning message control options for stitching (all options are global).

spferrorreport

Spectre Syntax

```
usim_opt spferrorreport=0|1|2
```

SPICE Syntax

```
.usim_opt spferrorreport=0|1|2
```

Description

Use this option to specify the level of error reporting.

Arguments

`0`	Reporting is turned off
`1`	Only error and warning messages are printed in the file (default)
`2`	All messages are printed in the file

Example

Spectre Syntax:

```
usim_opt spferrorreport=0
```

SPICE Syntax:

```
.usim_opt spferrorreport=0
```

tells the Virtuoso UltraSim simulator not to report any information (error, warning, or information messages).

spfmaxerrormsg

Spectre Syntax

```
usim_opt spfmaxerrormsg=value
```

SPICE Syntax

```
.usim_opt spfmaxerrormsg=value
```

Description

This option specifies the maximum number of backannotation errors reported by the simulator (default is 50).

Example

Spectre Syntax:

```
usim_opt spfmaxerrormsg=1000
```

SPICE Syntax:

```
.usim_opt spfmaxerrormsg=1000
```

tells the Virtuoso UltraSim simulator to print out up to 1,000 error messages.

spfmaxwarnmsg

Spectre Syntax

```
usim_opt spfmaxwarnmsg=value
```

SPICE Syntax

```
.usim_opt spfmaxwarnmsg=value
```

Description

This option specifies the maximum number of warning messages reported during stitching (default is 50).

Example

Spectre Syntax:

```
usim_opt spfmaxwarngmsg=1000
```

SPICE Syntax:

```
.usim_opt spfmaxwarngmsg=1000
```

tells the Virtuoso UltraSim simulator to print out up to 1,000 warning messages.

Stitching Statistical Reports

The Virtuoso UltraSim simulator reports stitching statistics in a log file (for example, how many resistors or capacitors are stitched). The following is an example of a stitching report:

```
Reading SPF file ./ring_top.spef line 32 ( 2.5% )
Reading SPF file ./ring_top.spef line 160 (19.7% )
Reading SPF file ./ring_top.spef line 285 (36.6% )
Reading SPF file ./ring_top.spef line 405 (51.2% )
Reading SPF file ./ring_top.spef line 533 (68.5% )
Reading SPF file ./ring_top.spef line 658 (85.4% )
Reading SPF file ./ring_top.spef line 777 (100.0% )

SPF Parsing: user time: 0:00:00 (0.010 sec), system time: 0:00:00 (0.000 sec),
real time: 0:00:00 (0.010 sec)
SPF Parsing: memory:                  0 B   total:       15.2684 MB
```

```
SPF Collect nets: user time: 0:00:00 (0.000 sec), system time: 0:00:00 (0.000 sec),
real time: 0:00:00 (0.000 sec)
SPF Collect nets: memory:             0 B   total:       15.2684 MB
Back annotation: user time: 0:00:00 (0.000 sec), system time: 0:00:00 (0.000 sec),
real time: 0:00:00 (0.000 sec)
Back annotation: memory:              131.0399 KB   total:       15.3994 MB

---------------------------------------------------------------------

 Nets      | parsed         6|    expanded         6|   errors         0
 Capacitors| parsed       485|    expanded       141|   stitch        70
 Resistors | parsed       110|    expanded        98|   stitch        49
 New nodes | added         90|                      | net coll         3

---------------------------------------------------------------------

 nets stitched with C-only 0        nets stitched with RC 6

---------------------------------------------------------------------

 0 errors and 13 warnings are issued (see file "my_input.spef.spfrpt")

---------------------------------------------------------------------

Messages statistics

=========================

Shortened R/C excluded               6
Dangling R/C terminal                12
Subckt is detected with top path     1
Instance is duplicated or fingered   6

Stitching Maximum Memory usage 15399456

TotalFlatRBefore      = 0
TotalFlatCBefore      = 0
TotalFlatRAfter       = 98
TotalFlatCAfter       = 140
TotalHierRBefore      = 0
TotalHierCBefore      = 0
TotalHierRAfter       = 49
TotalHierCAfter       = 70
CircuitCountFlatBefore= 10
CircuitCountHierBefore= 4
BranchCountFlatBefore = 4
BranchCountHierBefore = 3
LeafCountFlatBefore   = 6
```

```
LeafCountHierBefore   = 1
CircuitCountFlatAfter = 16
CircuitCountHierAfter = 9
BranchCountFlatAfter  = 4
BranchCountHierAfter  = 3
LeafCountFlatAfter    = 12
LeafCountHierAfter    = 6
StitchedNetsFlat      = 6
StitchedNetsHier      = 3
NetParsed             = 6
NetExpandeded         = 6
NetStitched           = 3
NetStitchRatio        = 100.00%
NetCompressRatio      = 50.00%
Rparsed               = 110
Rexpanded             = 98
Rstitched             = 49
RStitchRatio          = 89.00%
RCompressRatio        = 50.00%
CgParsed              = 92
CgExpanded            = 141
CgStitched            = 70
CgStitchRatio         = 153.00%
CgCompressRatio       = 49.00%
CcParsed              = 393
CcExpanded            = 0
CcStitched            = 0
CcStitchRatio         = 0.00%
CcCompressRatio       = 0.00%
HierNodesBefore       = 7
FlatNodesBefore       = 9
HierNodesAfter        = 44
FlatNodesAfter        = 83
AverageReduction      = 100.0
Nets(RC<100)          = 6
NetsH(RC<100)         = 3
NetsHR(RC<100)        = 3
Splitted ckts         = 2
Created RC ckts       = 3
Total/merged devpaths = 18/0
```

Frequently Asked Questions

How can I minimize memory consumption?

When simulating a flat post-layout design, you may run out of memory. To minimize memory consumption, use one of the following Virtuoso UltraSim simulator options.

keepparaname

Spectre Syntax

```
usim_opt keepparaname=0|1
```

SPICE Syntax

```
.usim_opt keepparaname=0|1
```

Description

The `keepparaname` option allows you to choose between preserving or not preserving the names of RC elements during parsing (default is 1). The option reduces memory usage when the names are not preserved and is only applicable to designed RCs (not parasitic RCs if the parasitics are stitched). If you set `usim_opt keepparaname=0`, the names of RC elements are not saved, in order to reduce memory usage.

Note: You cannot probe current through an element if this setting is used.

Since `keepparaname` is applied during netlist file parsing, it needs to be specified before reading a netlist file. You can also switch the settings of `keepparaname` for different segments of the netlist file.

Example

Spectre Syntax:

```
usim_opt keepparaname=0
usim_opt keepparaname=1
```

SPICE Syntax:

```
.usim_opt keepparaname=0
.usim_opt keepparaname=1
```

tells the Virtuoso UltraSim simulator not to save elements after the `usim_opt keepparaname=0` statement and before the `usim_opt keepparaname=1` statement (names are saved for the elements that follow `usim_opt keepparaname=1`).

cgndparse

Spectre Syntax

```
usim_opt cgndparse=value
```

SPICE Syntax

```
.usim_opt cgndparse=value
```

Description

Coupling capacitors less than `cgndparse` are grounded (default is 1e -18). `cgndparse` needs to be specified before reading a netlist file and can only be used in conjunction with `usim_opt keepparaname=0` (see keepparaname for more information).

How can I reduce the time it takes to run a DC simulation?

DC simulation can be time consuming for post-layout. To reduce the simulation time, you can save the pre-layout DC using `usim_save` and simulate the post-layout by starting DC simulation with `usim_restart` (Spectre syntax). This method has shown to reduce the post-layout DC simulation time significantly. The Virtuoso UltraSim simulator saves the information for all the external nodes in a file. For all internal nodes and states, if the nodes and states are registered in the solver, the information is also saved (otherwise, the simulator does not save the information in the file, and when the simulation is restarted, the unsaved internal nodes and states need to be solved again).

5

Voltage Regulator Simulation

This chapter describes how to perform voltage regulator (VR) simulations using the Virtuoso® UltraSim™ simulator.

Overview of Voltage Regulator Simulation

Due to the continuous reduction of supply voltage and the adoption of multiple supply voltages within a semiconductor chip, an increasing number of mixed signal/RF or digital circuits use on-chip voltage regulators to generate internal supply voltages.

Fast SPICE simulators depend on efficient partitioning to achieve simulation speed-up, which is only possible when the circuits are driven by an ideal power supply. Using conventional partition technology, all the blocks connected to an internally regulated supply have to be contained in a single partition, resulting in unacceptable simulation performance. VR simulation overcomes this limitation, enabling you to simulate designs with large circuit blocks powered by internal voltage regulators.

The conventional Virtuoso UltraSim simulator partitioning process, using `ms`, `da`, or `df` mode, is based on ideal supply voltages, such as dc or pwl (see "Simulation Modes and Accuracy Settings" on page 157 for more information about Virtuoso UltraSim simulation modes). If a design is powered by one or multiple VRs, or the supply voltage is generated by other source types (for example, controlled sources or sinus type), the Virtuoso UltraSim partitioning approach may result in large partitions and decreased simulation performance. Cadence recommends using VR simulation to resolve the performance issues produced by these applications. VR simulation is specifically designed for large mixed-signal, digital, and memory designs which are driven by regulators or other sources.

VR simulation is not applicable to pure power management blocks or designs with sensitive coupling between the generator and the driven circuit. Cadence recommends using `a` mode for these applications.

You need to identify the internal supply voltage nodes and driving blocks before using VR simulation. Supply voltage nodes are characterized by a large number of channel connections (CC). Use the Virtuoso UltraSim simulator `usim_report node` option to identify supply voltage nodes (see "Node Connectivity Report" on page 516 for more information).

usim_vr

Spectre Syntax

```
usim_vr inst=[inst1 inst2 ...] node=[node1 node2 ...]
usim_vr subckt=subckt1 node=[node1 node2 ...]
usim_vr subckt=subckt1 port=[port1 port2 ...]
```

SPICE Syntax

```
.usim_vr inst=[inst1 inst2 ...] node=[node1 node2 ...]
.usim_vr subckt=subckt1 node=[node1 node2 ...]
.usim_vr subckt=subckt1 port=[port1 port2 ...]
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.usim_vr`).

Description

Use `usim_vr` to run a Virtuoso UltraSim VR simulation (only applicable to circuits simulated in `df`, `da`, or `ms` mode). A netlist file can contain multiple `usim_vr` commands. VR simulation produces optimal results with a strong regulator driving capacitive load and weak dc loading. Circuits with on-chip voltage regulators driving large digital blocks generally belong to this category. Table 5-1 on page 322 contains all of the `usim_vr` commands and descriptions.

Note: You can also perform VR simulations in the Virtuoso Analog Design Environment (ADE). For more information, refer to "Setting Voltage Regulator Simulation Options" in the *Virtuoso Analog Design Environment L User Guide* (IC 6.1.2) or the *Virtuoso Analog Design Environment User Guide* (IC 5.1.41).

Table 5-1 usim_vr Commands

Command	Description
inst1, inst2, ...	Specifies the instances of the voltage regulator blocks (multiple regulators can be defined). Cadence recommends that all circuit blocks which contribute to the generation of the regulated supply voltages need to be specified by multiple block arguments.

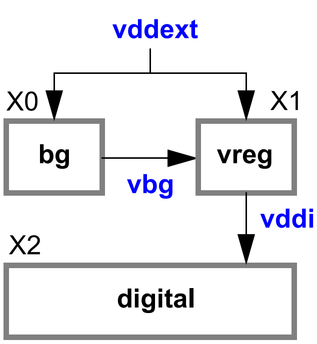
Table 5-1 usim_vr Commands, *continued*

Command	Description
subckt1	Specifies the subcircuit of the voltage regulator.
node1, node2, ...	Specifies the full hierarchical name of the internal power supply nodes driven by the voltage regulator (multiple regulated supply nodes can be specified).
port1, port2, ...	Specifies the port name of the internal power supply nodes driven by the voltage regulator (multiple ports can be specified). The ports are defined in the port list of subckt1.

Note: The Virtuoso UltraSim simulator supports using wildcards (*) in instance, node, and port names.

Examples

Bandgap Reference with Supply Voltage Generator



Spectre Syntax:

```
usim_opt sim_mode=df
usim_opt sim_mode=ms subckt=[bg vreg]
usim_vr subckt=bg
usim_vr subckt=vreg node=[vddi]
```

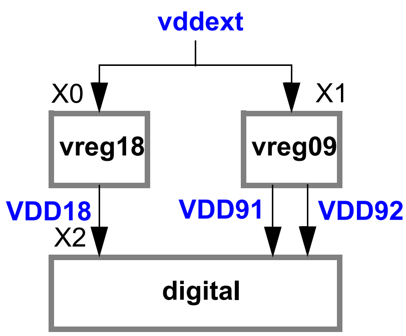
SPICE Syntax:

```
.usim_opt sim_mode=df
.usim_opt sim_mode=ms subckt=[bg vreg]
.usim_vr subckt=bg
```

```
.usim_vr subckt=vreg node=[vddi]
```

In this example, the voltage regulator consists of a bandgap reference generator (`bg`) and the supply generator (`vreg`), with an internally regulated supply node (`vddi`). The Virtuoso UltraSim simulator `usim_vr` command specifies the generator blocks and internal supply nodes. Global `df` mode is used to simulate the circuit, and local `ms` mode is applied to the `bg` and `vreg` blocks.

Multiple Generator Blocks



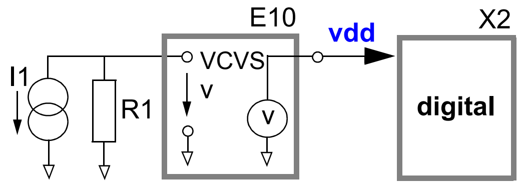

```
usim_opt sim_mode=df inst=[X2]
usim_vr inst=[X0 X1] node=[VDD18 VDD91 VDD92]
```

or

```
usim_opt sim_mode=df inst=X2]
usim_vr inst=[X0] node=[VDD18]
usim_vr inst=[X1] node=[VDD91 VDD92]
```

In this example, the design contains multiple generator blocks, and the internally regulated supply nodes are `VDD18`, `VDD91`, and `VDD92`. The `usim_vr` command specifies that the `X0` and `X1` instances are the voltage regulator blocks, global `ms` mode is used to simulate the circuit (default), and local `df` mode is applied to the digital block.

Voltage Supply Generated by Controlled Source

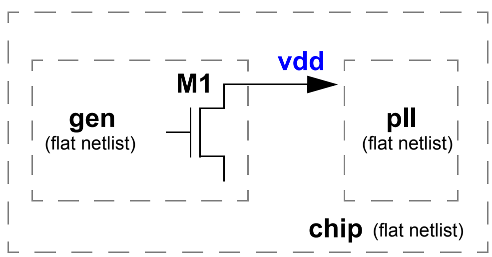



```
usim_opt sim_mode=df [X2]
usim_vr inst=[E10 R1 I1] node=[vdd]
```

In this example, the voltage supply is generated by a controlled source, and the internally regulated supply node is `vdd`. The `usim_vr` command specifies the `E10`, `I1`, and `R1` elements as part of the voltage regulator, global `ms` mode is used to simulate the circuit (default), and local `df` mode is applied to the digital block. The `usim_vr` arguments can be used in any order.

Note: You have the option to specify only one of the elements in the statement (`E10`, `I1`, or `R1`).

Design with Flat Netlist File




```
usim_vr inst=[M1] node=[vdd]
```

In this example, the design is a flat netlist file without subcircuit definitions. Since there are no subcircuit blocks or instances in a flat netlist file, the block option cannot be specified. To setup a VR simulation, you only need to specify one element that is part of the generator block (in this case, `M1` as the transistor and `vdd` as the generator node). The simulator uses this information to automatically identify the block.

6

Power Network Solver

This chapter describes how to detect and analyze power networks using the Virtuoso® UltraSim™ power network solver (UPS).

Detecting and Analyzing Power Networks

The Virtuoso UltraSim simulator can be set to detect power networks. A power network is a RC network that is driven by a power supply or internal generator, and it sends voltage through channel connections into a large number of active devices (for example, a MOSFET device). In most cases, the power network is used to model the IR drop in power supplies, yet power networks can also appear in signal nets.

The methods of detection include:

- **Automatic** – simulator automatically detects the power network according to its connectivity. Use the pn_level and pn_max_res methods to get finer grid control for automatic detection.
- **Manual** – you can manually specify a substring for a power net name (known as string or name mapping). Only the elements, primarily resistors and capacitors with terminal names that contain the specified substring, are kept in the power networks. Use the usim_pn command to specify the pattern and `pn_max_res` to further control partitioning.

usim_pn

Spectre Syntax

```
usim_pn node=name <pattern=[pattern1, pattern2, ...]> <method=short|keep|ups>
usim_pn auto=yes|no <method=short|keep|ups>
```

SPICE Syntax

```
.usim_pn node=name <pattern=[pattern1, pattern2, ...]> <method=short|keep|ups>
.usim_pn auto=yes|no <method=short|keep|ups>
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.usim_pn`).

Description

The `usim_pn` command is used to specify how power network nodes are detected and handled by the Virtuoso UltraSim simulator.

Arguments

`node`	Name of the power network node (wildcards are not supported).
`pattern1`, *`pattern2`*	Strings for pattern matching. Only nodes with names containing *`pattern1`* or *`pattern2`* are partitioned into power networks.
`method`	Method applied to the nodes previously specified – determines how parser networks are handled. ■ `method=short` removes the power network (default for `ms`, `da`, and `df` modes) ■ `method=keep` keeps the power network and simulates it with the circuit ■ `method=ups` keeps the power network and analyzes it with the UPS solver (rest of the circuit is simulated by the simulator)
`auto`	Auto-detection mode for the power network nodes. ■ `auto=yes` enables auto-detection (default for `ms`, `da`, and `df` modes) ■ `auto=no` disables auto-detection (default for `s` and `a` modes) **Note:** You need to specify nodes manually.

Example

Spectre Syntax:

```
usim_pn node=vdd method=keep
usim_pn node=gnd method=short
usim_pn auto=yes method=ups
```

SPICE Syntax:

```
.usim_pn node=vdd method=keep
.usim_pn node=gnd method=short
.usim_pn auto=yes method=ups
```

tells the Virtuoso UltraSim simulator to keep the `vdd` power network, simulates the network, removes the `gnd` power network, and uses auto-detection to find all of the other power networks to which the simulator applies UPS.

pn_level

```
usim_opt pn_level=0|1|2|3|4
```

Description

The Virtuoso UltraSim simulator uses a detection algorithm to automatically detect power networks. The `pn_level` method is used to control the aggressiveness of the power network detection algorithm. The higher `pn_level` is set (range of `0` to `4`), the more aggressive the detection algorithm, which results in more nets being classified as power nets. The default is `pn_level=0`.

Note: This method only applies to automatic detection of power networks.

Example

```
usim_opt pn_level=4
```

tells the simulator to use the most aggressive simulation setting (`pn_level=4`) to automatically detect power networks.

pn_max_res

```
usim_opt pn_max_res=value
```

Description

Controls partitioning between signal and power nets via resistor values. Use `pn_max_res` to specify the resistor values. Resistors with a value less than the specified value are considered part of the power network, whereas resistors with a value equal to or larger than the specified value are part of the signal net.

Example

```
usim_opt pn_max_res=1000
```

tells the Virtuoso UltraSim simulator to partition resistors with a value less than 1000 ohms that are part of a specific power net.

pn

```
usim_opt pn=0 inst=[res1 res2 cap1 cap2...] model=[model1 model2..]
     pn_file=[“filename1” “filename2”...]
```

Description

The `pn=0` command is used to exclude resistors and capacitors from power network detection.

Table 6-1 pn=0 Options

Option	Description
`res1`, `res2`	Specifies the resistors to be excluded from power network detection (must use full hierarchical name).
`cap1`, `cap2`	Specifies the capacitors to be excluded from power network detection (must use full hierarchical name).
`filename1, filename2...`	The name of the files which contain the resistors and capacitors to be excluded (full hierarchical name for resistors and capacitors needs to be included in files).
`model1`, `model2`	Specifies the model names. The model names need to use the resistor or capacitor model names. All instances for the specified model are excluded from power network detection.

Example

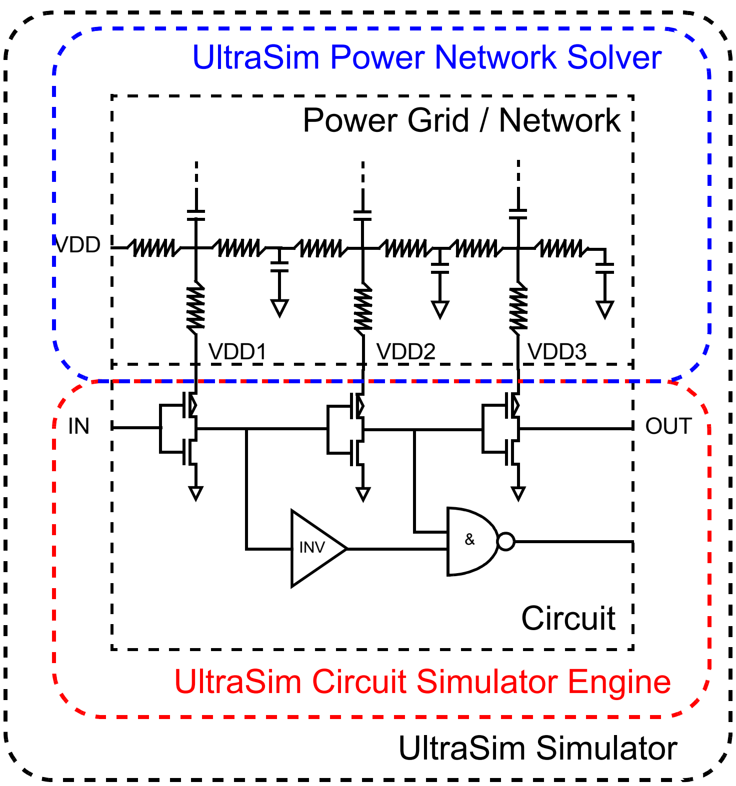
```
usim_opt pn=0 inst=[x1.r1 x1.c2]
```

tells the simulator to exclude the `x1.r1` and `x1.c2` resistors from any power networks, even if the resistors share connectivity.

UltraSim Power Network Solver

The UPS is an optimized solver designed to analyze linear power networks. The solver is integrated into the Virtuoso UltraSim simulator, and together with the Virtuoso UltraSim engine, lets you calculate the IR drop in power networks and analyze its effect on circuit behavior. Figure 6-1 shows an overview of the power network simulation methodology recommended by Cadence.

Figure 6-1 Overview of Power Network Simulation Methodology



The `method=ups` argument in the usim_pn command is used to enable UPS. Additional power network solver options can be set using the `usim_ups` keyword (see the table of "Arguments" on page 332 for a list of `usim_ups` arguments).

usim_ups

Spectre Syntax

```
usim_ups <iteration=integer> <speed=number> <ir_avg_threshold=double>
     <ir_peak_threshold=double> <ir_rms_threshold=double>
     <ir_report=filename> <waveform_file=filename>
     <all_waveform=true|false>|<output_node_file=filename>>
```

SPICE Syntax

```
.usim_ups <iteration=integer> <speed=number> <ir_avg_threshold=double>
     <ir_peak_threshold=double> <ir_rms_threshold=double>
     <ir_report=filename> <waveform_file=filename>
     <all_waveform=true|false>|<output_node_file=filename>>
```

Arguments

`iteration`	Number of iterations between UPS and the Virtuoso UltraSim engine. The higher the iteration number, the greater the accuracy (default is `iteration=1`). The `iteration` argument only provides IR drop and does not calculate its influence on circuit behavior. To achieve higher accuracy for a large IR drop, Cadence recommends using `iteration=2` or greater.
`speed`	Designates the speed and accuracy trade-off for UPS, with speed levels ranging from 1 to 8. The speed settings are as follows (default is `speed=3`): ■ **speed=1** lowest speed, highest accuracy ■ **speed=2** through **speed=7** moderate speed and accuracy ■ **speed=8** highest speed, lowest accuracy
`ir_avg_threshold`	Threshold average for IR drop reporting (default print out is 20 nodes with highest average IR drop).
`ir_peak_threshold`	Threshold peak for IR drop reporting (default print out is 20 nodes with highest peak IR drop).
`ir_rms_threshold`	Threshold RMS for IR drop reporting (default print out is 20 nodes with highest average RMS drop).

`ir_report`	Filename to which IR report is printed (default is `none`).
`waveform_file`	Filename of waveform file containing voltages for selected nodes, power networks, and tap points (default is `none`).
`all_waveform`	Prints out all power network node voltages (`true`) or the tap point connected to active devices (`false`). Default is `false`. **Note:** You first need to define `waveform_file` before `all_waveform` is enabled.
`output_node_file`	Filename that contains the nodes from the waveform file. Allows you to choose the nodes of interest.

Note: The UltraSim simulator only accepts one `usim_ups` statement at a time. If multiple `usim_ups` statements are specified in the netlist file, the simulator uses the last statement and ignores the rest.

Example

Spectre Syntax:

```
usim_pn node=VSS_D pattern=[VSS_D] method=ups
usim_pn node=VSS_A pattern=[vss_a] method=ups
usim_ups iteration=1 speed=2 waveform_file=wave all_waveform=true
usim_opt pn_res_max=50
```

SPICE Syntax:

```
.usim_pn node=VSS_D pattern=[VSS_D] method=ups
.usim_pn node=VSS_A pattern=[vss_a] method=ups
.usim_ups iteration=1 speed=2 waveform_file=wave all_waveform=true
.usim_opt pn_res_max=50
```

tells the Virtuoso UltraSim simulator:

- To partition power nets according to the name mapping mechanism.

  For nodes associated with power net `VSS_D`, the simulator only partitions elements with terminal names containing `VSS_D` to be solved by UPS. For nodes associated with `VSS_A`, the simulator only partitions elements with terminal names containing `vss_a` to be solved by UPS. For nodes associated with `VSS_D`, the simulator only partitions elements with terminal names containing `VSS_D` to be solved by UPS.

- The iteration number between the simulator engine and UPS is `1` (default), and speed is `2` (moderate speed and accuracy).

- All waveform files start with `wave` and they contain all of the power network nodes.
- To limit the maximum resistance in the power nets to 50 ohms.

7

Interactive Simulation Debugging

Overview of Interactive Simulation Debugging

The Virtuoso® UltraSim™ simulator interactive circuit debugging mode allows you to obtain design data, such as circuit elements and parameters, circuit topology, and instantaneous signal values. It can also be used to probe dynamic circuit behavior, including voltage and current waveforms simulated to the current time step.

The `UltraSim[I]*>` prompt is displayed (`I` is an integer) in interactive mode, indicating the Virtuoso UltraSim simulator is ready to receive interactive mode commands. The interactive commands shell allows you to apply any Tcl commands and constructs, and to redirect output of interactive shell commands to a file. All system commands supported by Tcl are also supported in the interactive mode.

To invoke the Virtuoso UltraSim simulator interactive mode:

- Use `-i` in the command line
- Use `-cmd` *`<command_file_name>`* in the command line
- Type `Ctrl-C` at any time after DC initialization

You can generate different types of output files in interactive mode:

`netlist.icmd`	Lists the history of all Virtuoso UltraSim simulator commands used.
`netlist.ilog`	Contains a list of commands and simulator outputs (copy of `stdout` file). **Note:** The log file can be opened and closed during interactive mode, but only one log file is active at a time.

General Commands

The Virtuoso UltraSim simulator supports the following interactive simulation general commands:

- alias on page 337
- exec on page 338
- exit on page 339
- help on page 340
- history on page 341
- runcmd on page 342

alias

`alias [`*`alias_name`*` [`*`cmd_name`*`]]`

Description

This command is used to create the alias name (`alias_name`) for the existing interactive command `cmd_name`. If the command is used without arguments, the existing list of aliases is printed.

Note: Existing interactive mode commands cannot be used as an alias name.

Caution

Using this command can overwrite the existing alias name (no warnings are generated).

Examples

For example

```
alias openlog open
```

tells the Virtuoso UltraSim simulator to create an `openlog` alias for the interactive mode command `open`.

In the next example

```
alias
```

provides a list of existing aliases.

In the next example

```
alias openlog
```

prints the command aliased to `openlog`.

exec

`exec` *`unix_shell_command`*

Description

Allows you to execute any UNIX or Linux command.

Example

```
exec ls
```

Lists all of the files in the current working directory.

exit

```
exit
```

Description

This command is used to stop the Virtuoso UltraSim simulator and exit the simulation.

Example

```
UltraSim[1]*> exit
    Log file "./simulation.ilog" closed
```

tells the simulator to end the simulation and exit (interactive log file is automatically closed).

help

`help [`*`cmd_name`*`|-all]`

Description

Use this command to display the `cmd_name` syntax and description. If `cmd_name` is not specified, a list of all interactive commands is printed. If the `-all` option is specified, the syntax for all interactive commands is printed.

Example

```
help read
```

tells the simulator to display the `read` command syntax and description.

history

`history`

Description

The `history` or `h` command prints out a list of the last 30 interactive commands used. You can use `!!` to recall the last command in the interactive shell. To recall a previously used interactive command with an index, use `!index`.

Example

```
UltraSim[1]*> history
    1 history
UltraSim[2]*> flush
UltraSim[3]*> !!
    flush
UltraSim[4]*>!1
    history
        1 history
        2 flush
        3 flush
        4 history
```

tells the simulator that `history` is the first Virtuoso UltraSim command (section [1]), flush is the second command (section [2]), and `!!` prints out the last command used (section [3]).

Note: Use `!1` to print out the first command in the history list.

runcmd

`runcmd [-v] *cmd_file*`

Description

The `run_cmd` command reads the `cmd_file` and executes the sequence of interactive mode commands defined in the file. Multiple command files can be used, but `run` and `stop` commands are only executed from the main file (main file is the command file called from the Virtuoso UltraSim simulator command line or first read in by the `runcmd` file). To display the commands, set the `-v` option.

Example

```
runcmd chip.icmd
```

tells the simulator to read and execute interactive commands from the `chip.icmd` file.

Log File Commands

The Virtuoso UltraSim simulator supports the following interactive simulation log file commands:

close

```
close
```

Description

This command is used to close the active log file.

Example

```
UltraSim[1]*> close
    Log file "./simulation.ilog" closed
```

tells the simulator to close the active log file.

flush

```
flush
```

Description

Use this command to flush all of the waveform data, as well as the log file information, into related output files for the current time.

Example

```
UltraSim[1]*>flush
```

tells the simulator to flush or move all of the waveform and log file data for the current time point into related output files.

open

`open [-a] logfile_name`

Description

Opens the `logfile_name` file in the current working directory in which interactive mode commands and results can be recorded (Tcl interface and log file outputs are the same). If the `-a` option is set, the print out is appended to the existing log file (otherwise the existing log file is overwritten).

Note: Using this command automatically closes the previous log file.

Example

```
open chip.log
```

tells the simulator to open the `chip.log` file and writes all interactive mode information into the file.

Analysis Commands

The Virtuoso UltraSim simulator supports the following interactive simulation analysis commands:

conn

`conn -n node_name|-ni node_index [-level 0|1] [-num value]`

Description

This command is used to report connectivity information for the node specified by name or node index. The amount of information reported is controlled by `level` and `num` is used to dump elements (default=50). The `conn` command is generally used in conjunction with the describe and node commands to trace connectivity.

The `level` argument consists of:

- **level 0** generates a summary of elements connected to the node, number of R/C/MOS/ DIODE devices, and a description of each device.
- **level 1** generates a summary of elements connected to the node, number of R/C/MOS/ DIODE devices, and shows all terminals.

Examples

In the following example

```
conn -n wl<5> -level 0
```

generates the following `level 0` summary:

```
RES: Name=xpost0.xi0.xi5.xinv3.$#R43.$#R43_0, ID=0, res=1.10887
*                  NODE0:                   wl<5>
RES: Name=xpost0.xi0.xi5.xinv3.$#R43.$#R43_1, ID=1, res=22.7632
*                  NODE0:                   wl<5>
CAP: Name=xpost0.xi0.xi5.xinv3.$#R43.$#R43_4, ID=6, cap=9.37962e-16
*                  NODE0:                   wl<5>
Summary of devices connected to node wl<5>
2 Resistors, 1 Capacitors
```

In the next example

```
conn -n wl<5> -level 1
```

generates the following `level 1` summary:

```
RES: Name=xpost0.xi0.xi5.xinv3.$#R43.$#R43_0, ID=0, res=1.10887
*                  NODE0:                   wl<5>
                   NODE1:                   xpost0.xi0.xi5.xinv3.$#R43.$#R43_4
RES: Name=xpost0.xi0.xi5.xinv3.$#R43.$#R43_1, ID=1, res=22.7632
*                  NODE0:                   wl<5>
```

```
                   NODE1:                   xpost0.xi0.xi5.xinv3.$#2
CAP: Name=xpost0.xi0.xi5.xinv3.$#R43.$#R43_4, ID=6, cap=9.37962e-16
*                  NODE0:                   wl<5>
                   NODE1:                   0
Summary of devices connected to node wl<5>
2 Resistors, 1 Capacitors
3 Channel connected and 0 Non-channel connected elements
```

describe

`describe elem_name|-ei elem_index|-ii inst_index|-subckt subckt_name`

Description

The `describe` command is used to print detailed information for given element or instance names, including element type, parameters, terminals, terminal voltages, element currents, conductances, and capacitances. If `-ei elem_index` is used, detailed information for the element with the index of `elem_index` is printed. If `-ii inst_name` is used, the detailed information for the instance with the `instance_name` index is printed. If `-subckt subckt_name` is used, the elements included in the `subckt_name` block are printed.

Examples

In the first example

```
describe xpost0.xi0.xi5.xinv3.xp0.m0
```

prints the following information:

```
MOS: Name=xpost0.xi0.xi5.xinv3.xp0.m0, TYPE=pmos, ID=2
    + m = 1.000000e+00 ad = 1.756000e+00 as = 1.178000e+00 l = 1.300000e-01 pd =
    1.084800e+01
    + ps = 6.960000e+00 w = 6.200000e+00
    DRAIN:vdd voltage = 3
    GATE: xpost0.xi0.xi5.xinv3.$#0 voltage = 2.99998
    SOURCE: xpost0.xi0.xi5.xinv3.$#2 voltage = 7.19185e-05
BULK: vpb voltage = 3
```

In the next example

```
*> describe x1.xi134
```

tells the Virtuoso UltraSim simulator to output the following information:

```
Instance      x1.xi134       (opad_1)
       x1.0           0      voltage = 0
       x1.vdd!        vdd!   voltage = 5
       x1.net364      a      voltage = 2.49906
       x1.pa          y      voltage = 2.33685
```

In the next example

```
*> index -i x1.xi134
0
*> describe -ii 0
```

returns the following information:

```
Instance      x1.xi134       (opad_1)
       x1.0           0      voltage = 0
       x1.vdd!        vdd!   voltage = 5
       x1.net364      a      voltage = 2.49906
       x1.pa          y      voltage = 2.33685
```

In the next example

```
*> describe -subckt inv
```

returns the following information:

```
mm11
mm12
```

elem_i

`elem_i elem_name|-ei elem_index [-dc] [-term num_term]`

Description

The `elem_i` command is used to print the instantaneous current for all branches of the specified elements at the current simulation time, or return current through the terminal if specified. MOS transistors, resistors, capacitors, inductors, diodes, and bipolar junction transistors are some of the supported elements.

Arguments

`elem_name`	The elements for which the instantaneous currents of all branches are printed
`-ei`	The option precedes the index of elements
`elem_index`	The index of elements for which the instantaneous currents of all branches are printed
`-dc`	The option to print the static (dc) current
`-term`	The option to print the terminal current (precedes the terminal number)
`num_term`	The terminal number for which the current is printed

Examples

In the following example

```
elem_i  *.mm11
```

tells the Virtuoso UltraSim simulator to print the instantaneous current for all branches of the `*.mm11` MOSFET at the current simulation time, and generates the following report:

```
*> elem_i *.mm11
MOS: Name=x1.mm11, MODEL=pmos, ID=0
         + w = 4.800000e-06 l = 1.440000e-07 m = 1.000000e+00
         DRAIN:  current = -1.49252e-06
         GATE:   current = 6.28497e-06
         SOURCE: current = -3.02198e-06
         BULK:   current = -1.77047e-06
MOS: Name=x2.mm11, MODEL=pmos, ID=1
         + w = 4.800000e-06 l = 1.440000e-07 m = 1.000000e+00
         DRAIN:  current = 1.6687e-08
         GATE:   current = 2.38509e-06
```

```
         SOURCE: current = -2.36562e-06
         BULK:   current = -3.61571e-08
```

In the next example

```
*> elem_i x2.mm11 -term 1
2.385089e-06
*> elem_i x2.mm11 -term 2
-2.365619e-06
*> set drncurrent [elem_i x2.mm11 -term 0]
1.668699e-08
*> puts "drain current = $drncurrent"
drain current = 1.668699e-08
```

tells the simulator to return the terminal current (`-term` option is used).

In the next example

```
elem_i *.mm11 -dc
```

tells the simulator to return only the static (dc) current, and generates the following report

```
*> elem_i *.mm11 -dc
MOS: Name=x1.mm11, MODEL=pmos, ID=0
         + w = 4.800000e-06 l = 1.440000e-07 m = 1.000000e+00
         DRAIN:  current = 5.64378e-07
         GATE:   current = 0
         SOURCE: current = -5.64377e-07
         BULK:   current = -8.76415e-13
MOS: Name=x2.mm11, MODEL=pmos, ID=1
         + w = 4.800000e-06 l = 1.440000e-07 m = 1.000000e+00
         DRAIN:  current = 1.17393e-06
         GATE:   current = 0
         SOURCE: current = -1.17393e-06
         BULK:   current = -1.29182e-15
```

exi

```
exi [-ith threshold][-v] elem_name [elem_name]|-ei elem_index [elem_index]
```

Description

The `exi` command reports the elements for the specified element name that have a current greater than the specified threshold current value (in amperes). The default value is 5e-5A and wildcards (*) are supported. For more information about wildcards, see "Wildcard Rules" on page 55.

The results are printed either as a list of element names or indices. If `-v` is defined, the element indices are printed in a list called `usim_index_list` (if `-v` is not defined, the element names are printed). These lists are helpful when performing additional analyses on elements.

This command can also be used in a Tcl script. For example, the list can be reprinted using the `puts $usim_index_list` Tcl command.

Note: You can use the name command to find the element name that corresponds to an element index.

Examples

In the following example

```
exi –ith 1.0e-12 x1.m*
```

tells the Virtuoso UltraSim simulator to print out elements with the names that match `x1.m*` with a current greater than 1.0e-12A, and generates the following output:

```
UltraSim[3]*> exi -ith 1.0e-12 x1.m*
MOS: Name=x1.mm1, MODEL=pmos, ID=1 ElemKey(0x18e492a8), ModelCard(0x18b407b8)
        + w = 4.000000e-06 l = 1.800000e-07 m = 1.000000e+00
        DRAIN:  current = 4.68577e-12
        GATE:   current = 0
        SOURCE: current = -4.68577e-12
        BULK:   current = -3.10931e-21
MOS: Name=x1.mm12, MODEL=nmos, ID=2 ElemKey(0x18e49d58), ModelCard(0x18b406b8)
        + w = 2.000000e-06 l = 1.800000e-07 m = 1.000000e+00
        DRAIN:  current = -4.68597e-12
        GATE:   current = 0
        SOURCE: current = 2.88485e-12
        BULK:   current = 1.80068e-12
```

In the next example

```
exi –ith 1.0e-12 –v x1.*
```

tells the simulator to print out the `usim_index_list`, and generates the following output:

```
UltraSim[3]*> exi -ith 1.0e-12 –v x1.*
1 2
```

exitdc

```
exitdc
```

Description

This command is used to end a pseudo-transient simulation and start a transient simulation.

Example

```
UltraSim[1]*> exitdc
t=0.000000e+00 ...
```

tells the simulator to exit the dc analysis (warning message is also generated).

force

```
force (node_name [=] volt_value) ...|-ni (node_index [=] volt_value)...
     [-rt ramp_time]
```

Description

This command forces the node voltage of specified nodes to a specific voltage value. Nodes can be specified by name (`net_name1 net_name2 ...`) or index (`-ni index1 index2 ...`). Voltage is applied until released or the end of the simulation is reached. If several `force` statements are activated, the current one overwrites the previous commands.

Notes:

- The equal (`=`) sign is optional.
- The `force` operation is not performed if the net is connected to a static `vsource dc=0` and one vsource port is connected to the solver ground. One solution is to replace the static `vsource dc=0` with a PWL type vsource.
- To check if a net is connected to the static `vsource dc=0`, use the Virtuoso UltraSim `.usim_opt nodecut_file=1` option.

  Once the simulation starts, a `netlist.nodecut` file is created. Open the file and check the net in the node cut list. The net is shorted to `solver node 0`, so the `force` command is not applied to the net.
- To check if a specific force is applied during the simulation:
  - Start Virtuoso UltraSim interactive mode and use `force my_net1 2`.
  - Setup a short simulation time (for example, 1 ps) to run `force` in the solver.
  - Run `1p -relative` and check the values using `value my_net1`.

Examples

For example

```
force  xi0.xi3.gate net12 2.0
```

tells the Virtuoso UltraSim simulator to force v(xi0.xi3.gate) and v(net12) to 2.0 V.

In the next example

```
force -ni 113 1.05
```

tells the simulator to force nodes with index 113 to 1.05 V.

forcev

```
forcev vector logic_value [-rt ramp_time]
```

Description

This command is used to force nodes specified by the vector of node indices to a logic value.

Example

```
set add [index -n add<3> add<2> add<1> add<0>]
forcev $add " 'b0101"
```

tells the Virtuoso UltraSim simulator to create a vector (`set v`) and then forces `'b01X10` to the vector (`forcev`).

hier_tree

`hier_tree [-level num ] [ -a ] [ -def ] [ -num count] [-subckt name]`

Description

The `hier_tree` command is used to print the hierarchical tree for the specified subcircuit instances. If no subcircuit or instance name is specified, the Virtuoso UltraSim simulator starts from the top level of the design.

Arguments

`-level`	Limits the printed levels of hierarchy tree below the current level (default prints all levels).
`-a`	Prints the full hierarchical instance name (default prints only the local name).
`-def`	Prints the subcircuit name of the instance.
`-num`	Limits the number of printed instances (default is 50 instances).
`-subckt`	Specifies the instance from which the hierarchy tree is printed (default is `top` and wildcards are supported).
	For more information about wildcards, see "Wildcard Rules" on page 55.

Example

```
UltraSim[1]*>hier_tree -a
> xpost2(1)
> xpost2.xi0(2)
> xpost2.xi0.xi4(3)
> xpost2.xi0.xi4.xinv2(4)
> xpost2.xi0.xi4.xinv2.xn0(5)
> xpost2.xi0.xi4.xinv2.xp0(5)
…

UltraSim[2]*>hier_tree -a -def
> xpost2(decwl64b)
> xpost2.xi0(decwl8b)
> xpost2.xi0.xi4(trnoff)
> xpost2.xi0.xi4.xinv2(inv)
> xpost2.xi0.xi4.xinv2.xn0(nmos)
```

```
> xpost2.xi0.xi4.xinv2.xp0(pmos)

UltraSim[3]*>hier_tree -a -def -subckt xpost2.xi0
> xpost2.xi0.xi4(trnoff)
> xpost2.xi0.xi4.xinv2(inv)
> xpost2.xi0.xi4.xinv2.xn0(nmos)
> xpost2.xi0.xi4.xinv2.xp0(pmos)
…
```

tells the simulator to print the full hierarchical name for the levels defined by a number (section [1]), subcircuit name for each instance (section [2]), and the hierarchical tree for `xpost2.xi0` (section [3]).

index

`index -e elem_name|-n net_name|-i instance_name`

Description

This command provides an index for net, element, or instance names. If `-e elem_name` is used, the `elem_name` index is printed. If `-n net_name` is used, the index of the net with `net_name` is printed. If `-i instance_name` is used, the `instance_name` index is printed.

Examples

For example

```
index -e xpost0.xi0.xi5.xinv3.xp0.m0
```

tells the simulator to return an index for the `xpost0.xi0.xi5.xinv3.xp0.m0` element.

In the next example

```
index -n xi1.xi9.xi7.net12
```

returns an index for the `xi1.xi9.xi7.net12` net.

In the next example

```
*> index -i x1.xi134
0
```

returns an index for the `x1.xi134` instance.

match

`match [ -e pattern1] [ -n pattern]`

Description

This command is used to find all elements or nodes with names matching a specific pattern.

Arguments

`-e`	Prints the specified elements
`-n`	Prints the specified nodes

Example

```
UltraSim[1]*> match –e xpost0*
xpost0.xi7.xi1.xinv3.xn0.m0
xpost0.xi7.xi1.xinv3.xp0.m0
xpost0.xi7.xi2.xi81.xn0.m0
xpost0.xi7.xi2.xi81.xn1.m0
xpost0.xi7.xi2.xi81.xn2.m0
…
```

```
UltraSim[2]*> match –n xpost0.*
xpost0.xi7.xi7.xinv1.xp0.inh_vpb
xpost0.xi7.xi7.xinv1.xp0.s
xpost0.xi7.xi7.xinv2.xn0.d
xpost0.xi7.xi7.xinv2.xn0.g
xpost0.xi7.xi7.xinv2.xn0.inh_vnb
…
```

tells the simulator to print all elements and nodes with the `xpost0*` pattern.

meas

```
meas [ spice measurement statement ]
```

Description

The `meas` command is used to add measurements (SPICE syntax).

Example

```
UltraSim[1]*> meas tran period trig v(out) val=0.8 rise=2 targ v(out) val=0.8
rise=3
```

```
Succeed to set up the measure:
.meas tran tosc trig v(out) val=0.8 rise=2 targ v(out) val=0.8 rise=3
```

tells the simulator to use the `meas` command and measurement statement to obtain the period for the `out` signal.

name

`name [ -ei elem_index] [ -ni net_index] | -ii instance_index`

Description

Use this command to find the name of nets, elements, or instances by index. If `-ei elem_index` is used, the name of elements with `elem_index` is printed. If `-ni net_index` is used, the name of nets with `net_index` is printed. If `-ii instance_index` is used, the name of the instance with `instance_index` is printed.

Example

```
index -e xpost0.xi0.xi5.xinv3.xp0.m0
9
name -ei 9
xpost0.xi0.xi5.xinv3.xp0.m0
name -ni 5338
```

tells the Virtuoso UltraSim simulator to return the name of nets with net index 5338.

In the next example

```
*> name -ii 0
x1.xi134
```

tells the simulator to return the name of instance index 0.

nextelem

`nextelem`

Description

Allows you to individually view all elements in the circuit. You can use `nextelem 0` to view elements starting with the first one and `nextelem <index>` to view the elements directly.

Example

```
UltraSim[1]*> nextelem
name:v0 type:vsrc node0:vdd! node1:0

UltraSim[2]*> nextelem
name:c4 type:cap node0:net12 node1:0 capacitance:1.00000e-13

UltraSim[3]*> nextelem 0
name:v0 type:vsrc node0:vdd! node1:0

UltraSim[4]*> nextelem 3
name:xil5.mn type:nmos drn:net12 gate:net40 src:0 bulk:0 W:4.0000e-06 L:2.5000e-07
```

tells the simulator to retrieve the next element (sections [1] and [2]), restarts the iteration (section [3]), and displays the third element for viewing (section [4]).

node

`node net_name|-ni net_index`

Description

This command is used to print voltage information for given nets, including voltage, voltage slope (dv/dt), and node capacitance. Nets can be specified by name (*`net_name`*) or by index (`-ni` *`net_index`*).

Example

In the first example

```
node xpost0.xi7.xi7.xinv3.xp0.s
```

tells the Virtuoso UltraSim simulator to print node voltage and slope information, and generates the following output:

```
ode xpost0.xi7.xi7.xinv3.xp0.s : voltage = 7.16334e-05; (dV/dT) = 0
Ctot=1.037e-14
```

In the next example

```
node -ni 1
```

tells the simulator to perform an inquiry on the index of the specified node, and generates the following output:

```
node xpost0.xi7.xi7.xinv3.xp0.s : voltage = 7.16334e-05; (dV/dT) = 0
Ctot=1.345e-15
```

nodecon

`nodecon [ val ]`

Description

If `val` is specified, `nodecon` finds any nodes with connections greater than `val`. If `val` is not specified, `nodecon` finds nodes with the most connections, based on the netlist file. The report for each node contains the subcircuit name, node name, number of connections, and a `Vsrc` flag indicating if it is a voltage source.

Example

```
nodecon 100
0(3335)
 vpb(112)
 vnb(112)
 vss(3335)
 ad<7>(3335)
…
```

tells the Virtuoso UltraSim simulator to report any nodes with more than 100 connections.

op

```
op op_file
```

Description

This command is used to write the operating point (OP) at the current time to the `op_file`. If a pattern is specified, only the OP of nodes matching the pattern are printed.

Example

```
op chip.ic
```

tells the simulator to print the OP for all nets into the `chip.ic` file.

probe

`probe -n net_name1 net_name2 ...|-ni net_index1 net_index2 ...`

Description

The `probe` command is used to add analog waveform `v(`*`net_name1`*`)` and `v(`*`net_name2`*`)` to the waveform data list file. The added signal starts at the time it was added. You can also specify a node index using the `-ni` option.

Note: The `probe` command cannot be added when parameter storage format (PSF) is specified.

Examples

For example

```
probe -n fosc<1> fosc<2>
```

tells the simulator to add `fosc<1>` and `fosc<2>` to the fast signal database (FSDB) output file.

In the next example

```
probe -ni 255
```

adds the analog waveform for nets with index 255 to the FSDB file.

release

`release -all| net_name1  net_name2 ...|-ni net_index1 net_index2 ...`

Description

This command is used to release forced voltage from specific nodes. Nodes can be specified by name (`net_name1 net_name2 ...`) or by index (`-ni index1 index2 ...`).

Examples

For example

```
release xi0.xi3.gate net12
```

tells the Virtuoso UltraSim simulator to release v(xi0.xi3.gate) and v(net12).

In the next example

```
release -ni 113 105
```

tells the simulator to release nodes with index 113 and 105.

In the next example

```
release -ni $v
```

releases nodes specified by `$v` vector.

restart

`restart` *`filename`*

Description

The `restart` command allows you to load a previously saved intermediate state (save) and to continue the simulation starting at the saved time point. To restore the database results from a previously saved file (in netlist or interactive mode) and continue the simulation, use `filename`. When restarting the simulation from interactive mode at an earlier time, the output file contains an `.rs#` suffix with `#` representing the number of restarts.

Example

```
UltraSim[1]*> restart time@1.000000e-08
```

tells the simulator to restart the saved `time@1.000000e-08` file.

Note: The `time@1.000000e-08` file can be derived from `.usim_save` in the netlist file or from the save interactive command.

run

```
run [time [-absolute]|dtime -relative|numb_events  -absev|numb_events -relev]
```

Description

This command is used to continue the simulation. If arguments are not specified, the simulation continues until the next break point or the end of the simulation is reached. The `run` arguments can be used to specify the following actions:

- **-absolute** specifies the next break point by time, starting from the beginning of the simulation.
- **-relative** specifies the next break point from the current time.
- **-absev** and **-relev** specifies the next break point by the number of events (absolute or relative, respectively).

The next break point cannot occur before the current time or number of events (this condition is always true for relative arguments). If the break point is specified by a `stop` command or by any other means, and is closer than specified in the `run` command, the simulation stops at the nearest break point.

Examples

For example

```
run 25n
```

tells the Virtuoso UltraSim simulator to continue the simulation for 25 ns and then stop.

In the next example

```
run 5n -relative
```

tells the simulator to continue the simulation, starting from the current time, and stops after 5 ns.

In the next example

```
stop -create -time 20n -relative
run 30n -relative
```

continues the simulation, starting from the current time, and stops after 20 ns.

In the next example

```
run 10000 -relev
```

continues the simulation, starting from the current time, and stops after 1000 events are processed.

save

`save` *`filename`*

Description

The `save` command allows you to stop during the simulation and take a snapshot of the database. To save the simulation database at the current time to the `filename@time` file, use the `filename` argument.

Note: If `filename` is not specified, a file with the name *`design name`*`.save@time` is automatically generated.

Example

```
UltraSim[1]*> stop -create -time 10n

UltraSim[2]*> run

UltraSim[3]*> save time
Simulation state has been scheduled for the current time point.

UltraSim[4]*> run
```

tells the simulator to stop at transient time point 10 n (section [1]), continues the simulation (section [2]), saves the time point `time` (section [3]), and then continues the simulation to completion (section [4]). After the simulation ends, the `time@1.000000e-8` file is created.

Note: You can use the restart interactive command to continue the simulation by loading the saved file.

spfname

```
spfname -se element_spf_name | -sn node_spf_name | -ie element_internal_name | -in
     node_internal_nameDescription
```

Description

To save memory, Ultrasim uses compact names for stitched elements and nodes. The compact names follow the format `$#.$#RXXX`, such as in `$#.$#R0_0`. The `spfname` command prints the corresponding names as used in the DSPF/SPEF file and vice versa. To enable this command, set the following option in the netlist:

```
.usim_opt spfenablenameutil = 1
```

The default value for `spfenablenameutil` is `0`.

`-se element_spf_name`	The compact name for the element with the name `element_spf_name` in the DSPF/SPEF file is printed.
`-sn node_spf_name`	The compact name for the node with the name `node_spf_name` in DSPF/SPEF file is printed.
`-ie element_internal_name`	The name of the element whose compact name is `element_internal_name` is printed.
`-in node_internal_name`	The name of the node whose compact name is `node_internal_name` is printed.

Example

```
UltraSim[4]*> spfname -sn xi3852:n1
$#R0.$#R0_2
```

tells the simulator to print the compact name for the subnode `xi3852:n1` in the DSPF file.

stop

```
stop -create ((-time time|-event ev_number) ([-absolute]|-relative))|-delete
     breakpoint_index|-show [breakpoint_index ]
```

Description

This command is used to pause the simulation if any of the following conditions are met:

- **-show** prints a list of break points.
- **-delete** deletes a break point.
- **-create -time (time [absolute]|dtime –relative)** sets a break point at *<time|dtime>*.
- **-create -event (events [-absolute]|devents -relative)** sets a break point after *<events|devents>* events are triggered.

If arguments are not specified, all existing break points are automatically listed.

Example

```
stop -create -event 1000
1
```

```
stop -show
1 –ne 1000
```

```
stop -delete 15 29
```

tells the simulator to delete all break points with index=15 and 29.

time

`time [ -v]`

Description

The `time` command is used to print the current and final simulation time in seconds. If the `-v` option is specified, the current and end time are printed as a string. If unspecified, only the value of current time is printed, and it can be used as the Tcl variable.

Example

`time -v`

tells the simulator to print the current time=4.679620e-08 and end simulation time=1.000000e-07.

value

```
value [ format] node_name node_name ...|-ni node_index node_index...
```

Description

The `value` command prints the voltage or logic value of selected nodes. The `format` option supports the following values:

- **%g** (default) and **%e** – floating numbers
- **%d** – integer
- **%b** – binary
- **%o** – octal
- **%h** – hexadecimal

Formats `%e`, `%g`, and `%d` represent the analog node values and `%b`, `%o`, and `%h` the logic values. Logic values are calculated for the base of L0 and L1, where L0 = 0.0 v and L1 = 5.0 v (default). The default L0 and L1 values can be changed using `.usim_opt vh=`*`new value`* `vl=`*`new value`*.

Example

```
UltraSim[1]*> value 14
5.0198


UltraSim[2]*> value %b 14
‘b1


UltraSim[3]*> value %o 14
’o4


UltraSim[4]*> value %h 14
’h8


UltraSim[5]*> value %d 14
5


UltraSim[6]*> value %e 14
5.019798e+00
```

tells the simulator to find the value of node 14 for each format defined by `%b`, `%o`, `%h`, `%d`, and `%e`.

vni

```
vni [-threshold value]
```

Description

The `vni` command prints the voltage source node with a current greater than or equal to the threshold value (default is 0).

Example

```
vni –threshold 1.0e-12
```

tells the Virtuoso UltraSim simulator to print the voltage source node with a current greater than or equal to 1.0e-12 A, and generates the following output:

```
*> vni –threshold 1.0e-12
Vdd current=4.68577e-12
```

8

Virtuoso UltraSim Advanced Analysis

This chapter describes the following Virtuoso® UltraSim™ simulator advanced analysis methods, which include dynamic and static checks.

Dynamic Checks

- Active Node Checking detects nodes with voltage changes that exceed the user defined threshold.
- Design Checking monitors device voltages during simulation (device voltage check).
- Dynamic Power Checking reports the power consumed by each element and subcircuit in the design.
- Node Activity Analysis provides information about the nodes and monitors activity such as: voltage overshoots (VOs), voltage undershoots (VUs), maximum and minimum rise/fall times, switching activity, and half-swing flag.
- Power Analysis reports the average, maximum, and RMS current at the ports of specified subcircuits, child subcircuits, and grandchild subcircuits for a specified level of hierarchy.
- Wasted and Capacitive Current Analysis provides information about the capacitive, and static and dynamic wasted currents in specified subcircuits.
- Power Checking performs over current and high impedance node checks.
- Timing Analysis performs setup, hold, pulse width, and timing edge checks on signals.
- Bisection Timing Optimization combines multiple simulations into a single characterization.

Active Node Checking

The active node checking analysis detects nodes with voltage changes that exceed the user-defined threshold. With the active nodes identified, you can choose to selectively backannotate parasitic elements during post-layout simulation.

Spectre Syntax

```
acheck title node=[node1 node2...] <depth=value> dv=value <exclude=[node3
     node4...] time_window=[start1 stop1 start2 stop2...] <inactive=0|1|2>
```

SPICE Syntax

```
.acheck title node=[node1 node2 ...] <depth=value> dv=value <exclude=[node3
     node4...] time_window=[start1 stop1 start2 stop2...] <inactive=0|1|2>
```

or

Spectre Syntax

```
acheck dv=value
```

SPICE Syntax

```
.acheck dv=value
```

Notes:

- A period (`.`) is required when using SPICE language syntax (for example, `.acheck`).
- The `acheck dv=value` and `.acheck dv=value` syntax continues to be supported by Cadence (it has a higher capacity active node checking analysis for larger designs).

Description

This command is used to report the active nodes in a circuit design. A node is considered active if the change in its voltage exceeds `value` during the checking window. If a window is not specified, the entire simulation period is used. The active nodes are listed in `.actnode` and `.actnodelist` files. The inactive nodes are listed in `.inactnode` and `.inactnodelist` files. For the `.acheck dv=value` syntax, the nodes are reported in `.actnodelist` or `.inactnodelist` files.

Arguments

`title`	User-defined title name for the active node check.
`node1 node2 ...`	List of node names to be checked (wildcards are supported). For more information about wildcards, see "Wildcard Rules" on page 55.
`depth=value`	Defines the depth of the circuit hierarchy that a wildcard name applies to. If set to `1`, only the nodes at the current level are applied (default value is infinity).
`dv=value`	Defines the voltage change threshold for the active nodes (default is 0.1 volts).
`exclude`	Defines the node names to be excluded from the check (wildcards are supported).
`time_window`	Defines time period of checking.
`inactive=0|1|2`	Defines which nodes are reported. ■ `0` - only active nodes are reported (default) ■ `1` - only inactive nodes are reported ■ `2` - both active and inactive nodes are reported

Examples

In the following example

Spectre Syntax:

```
acheck dv=0.5
acheck achk1 node=[*] depth=2 dv=0.5 time_window=[10n 50n 100n 150n]
acheck achk2 node=[x1.*] dv=0.5 exclude=[x1.y1.* x1.y2.*]
acheck achk3 node=[x1.*] dv=0.5 exclude=[x1.y1.* x1.y2.*] inactive=1
```

SPICE Syntax:

```
.acheck dv=0.5
.acheck achk1 node=[*] depth=2 dv=0.5 time_window=[10n 50n 100n 150n]
.acheck achk2 node=[x1.*] dv=0.5 exclude=[x1.y1.* x1.y2.*]
.acheck achk3 node=[x1.*] dv=0.5 exclude=[x1.y1.* x1.y2.*] inactive=1
```

Design Checking

The Virtuoso UltraSim simulator allows you to perform a dynamic checking analysis on device voltages during a simulation by using the `dcheck` command. The analysis generates a report in a `.dcheck` file if the voltages exceed the specified voltage bounds.

To use the `dcheck` command, you can add it to the simulation netlist file, or place it in a command file and include the file in the simulation netlist file. The following design checking analyses are described in this section:

MOS Voltage Check

Spectre Syntax

```
dcheck title vmos topnode=0|1 <model=[model1, model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vgdl=volt> <vgdu=volt> <vdsl=volt> <vdsu=volt>
     <vdbl=volt> <vdbu=volt> <vgsl=volt> <vgsu=volt> <vgbl=volt> <vgbu=volt>
     <vsbl=volt> <vsbu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

SPICE Syntax

```
.dcheck title vmos topnode=0|1 <model=[model1, model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vgdl=volt> <vgdu=volt> <vdsl=volt> <vdsu=volt>
     <vdbl=volt> <vdbu=volt> <vgsl=volt> <vgsu=volt> <vgbl=volt> <vgbu=volt>
     <vsbl=volt> <vsbu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

Description

This command allows you to monitor metal oxide semiconductor (MOS) voltages during a simulation run, and generates a report if the voltages exceed the specified upper and lower bounds, or meets the specified conditions. You can exclude a subset of the instances from the voltage check using the `xsubckt` or `xinst` arguments. If a threshold or condition is not specified for `dcheck` in the netlist file, a warning message is issued by the Virtuoso UltraSim simulator and `dcheck` is ignored during the simulation.

Arguments

`title`	Title of the voltage check.
`topnode=0|1`	Specifies whether the top-level node name or the hierarchical terminal name is to be reported in the dcheck report file. If set to 0, the hierarchical device terminal name is reported (default). If set to 1, the top-level node name is reported. If a top-level node name is not available, the hierarchical terminal name is used.
`model`	MOS voltage check is applied to transistors matching the model name (wildcards are supported).
`subckt`	MOS voltage check is applied to transistors belonging to all instances of the subcircuits listed (wildcards are supported).
`xsubckt`	MOS voltage check is excluded from instances of the subcircuits listed (wildcards are supported).
`inst`	MOS voltage check is applied to transistors belonging to the subcircuit instances listed (wildcards are supported).
`xinst`	MOS voltage check is excluded from the subcircuit instances listed (wildcards are supported).
	Note: The `inst` and `xinst` arguments can only be used to specify subcircuit instances, but not device instances.
`vgdl=`*`volt`*	Reports the condition if Vgd is less than the specified lower bound voltage value.
`vgdu=`*`volt`*	Reports the condition if Vgd is greater than the specified upper bound voltage value.

`vdsl=`*`volt`*	Reports the condition if Vds is less than the specified lower bound voltage value.
`vdsu=`*`volt`*	Reports the condition if Vds is greater than the specified upper bound voltage value.
`vdbl=`*`volt`*	Reports the condition if Vdb is less than the specified lower bound voltage value.
`vdbu=`*`volt`*	Reports the condition if Vdb is greater than the specified upper bound voltage value.
`vgsl=`*`volt`*	Reports the condition if Vgs is less than the specified lower bound voltage value.
`vgsu=`*`volt`*	Reports the condition if Vgs is greater than the specified upper bound voltage value.
`vgbl=`*`volt`*	Reports the condition if Vgb is less than the specified lower bound voltage value.
`vgbu=`*`volt`*	Reports the condition if Vgb is greater than the specified upper bound voltage value.
`vsbl=`*`volt`*	Reports the condition if Vsb is less than the specified lower bound voltage value.
`vsbu=`*`volt`*	Reports the condition if Vsb is greater than the specified upper bound voltage. **Note:** The `vsbu` argument and other arguments listed in this table can be used with constant parameters, but must be enclosed by single quotation marks (for example, `vsbu='-par1'`).
`cond=`*`expression`*	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and variables: `vgs`, `vgd`, `vgd`, `vds`, `vdb`, `vsb`, `l`, `w`. The expression can be a combination of linear and non-linear expressions. The conditional check can be combined with the lower and upper threshold bounds mentioned in the Description. The conditional check (`cond=expr`) specifies which devices need to be checked and the threshold bounds (such as, `vgsl` and `vgsu`) are used to check if the devices contain violations.

`duration=`*`dtime`*	Reports the condition if device voltages are out of bounds for a duration of time longer than dtime (*`dtime`* default value is equal to the minimum time step of the simulation).
`time_window`	The time period specified for checking in which the first number is the start time point and the second number is the stop time point. For example, *`start1`* to *`stop1`* is the first time period and *`start2`* to *`stop2`* is the second time period. **Note:** Only ascending time points can be used (for example, *`start1`* < *`stop1`* < *`start2`* < *`stop2`*).
`probe=0|1`	Flag to probe node voltage for devices checked. If set to `0`, no probe is performed (default). If set to `1`, all node voltages for devices checked with `dcheck` are probed.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices

Examples

Spectre Syntax:

```
dcheck chk1 vmos model=[tt] inst=[X1] vgsu=1.0 vgsl=0.5 probe=1
dcheck chk2 vmos model=[tt2] cond=((vgs<-3 || vds>3) && l<0.2u)
dcheck chk3 vmos model=[tt3] cond=(vgs*vgs>1 && sin((2*3.14*vds)/0.45)>0.5 ||
vsb<-1) time_window=[1u 10u]
dcheck chk4 vmos model=[tt4] cond=(vgs>0.5 || vgd>0.5) vdsu=1.5
dcheck chk5 vmos xinst=[I2*] xsubckt=[Reg*] vgsu=1.86
dcheck chk6 vmos inst=[I19] xinst=[I19.I19 I19.I116] vgsu=1.86
dcheck chk7 vmos subckt=[pll*] xsubckt=[osc buf] vgsu=1.86
dcheck chk8 vmos subckt=[pll*] xsubckt=[osc buf] vgsu=1.86 topnode=1
```

SPICE Syntax:

```
.dcheck chk1 vmos model=[tt] inst=[X1] vgsu=1.0 vgsl=0.5 probe=1
.dcheck chk2 vmos model=[tt2] cond='(vgs<-3 || vds>3) && l<0.2u'
.dcheck chk3 vmos model=[tt3] cond='vgs*vgs>1 && sin((2*3.14*vds)/0.45)>0.5 ||
vsb<-1' time_window=[1u 10u]
.dcheck chk4 vmos model=[tt4] cond='vgs>0.5 || vgd>0.5' vdsu=1.5
.dcheck chk5 vmos xinst=[I2*] xsubckt=[Reg*] vgsu=1.86
.dcheck chk6 vmos inst=[I19] xinst=[I19.I19 I19.I116] vgsu=1.86
```

```
.dcheck chk7 vmos subckt=[pll*] xsubckt=[osc buf] vgsu=1.86
.dcheck chk8 vmos subckt=[pll*] xsubckt=[osc buf] vgsu=1.86 topnode=1
```

The command line of `chk1` checks all MOSFETs using model `tt` in block `X1`. The devices that meet `vgs>1` or `vgs<0.5` criteria are reported. Where `probe=1`, all node voltages of the `tt` devices are probed.

The command line of `chk2` checks all MOSFETs using model `tt2`, whether `vgs<-3` or `vds>3`, when MOSFET length is less than 0.2 um. If the condition is met, the devices are reported.

In the command line of `chk3`, a conditional design analysis check is performed by the simulator, and includes nonlinear expressions. The MOSFETs that meet the condition are reported.

The command line of `chk4` combines the conditional and upper/lower threshold bound checks. Only the MOSFET models that meet the specified conditions are checked (that is, MOSFET models are reported if vds>1.5).

In the command line of `chk5`, all MOSFETs in the netlist file are checked. The subcircuit instances with names that match `I2*`, instances of subcircuits with the name `Reg*`, and their sub-hierarchies are excluded. MOSFETs that meet the vgsu>1.86 criteria are reported.

In the command line of `chk6`, all MOSFETs belonging to instance `I19` and its sub-hierarchy are checked. Subcircuit instances `I19.I19` and `I19.I116` are excluded. MOSFETs that meet the vgsu>1.86 criteria are reported.

The command line of `chk7` checks all MOSFETs belonging to instances of the subcircuits with names that match `pll*` and their sub-hierarchies. Instances of subcircuits `osc` and `buf` are excluded. MOSFETs that meet the vgsu>1.86 criteria are reported.

The command line of `chk8` checks the same thing as `chk7` but reports the top-level node names rather than hierarchical terminal names into dcheck report file.

Sample Output

```
Title Model From  To    v_curr   chk_type Device   Drain Gate     Source   Bulk
            (ns)  (ns)  (vth)

chk1  tt    5.0   5.2   1.2      vgsu     X1.mn1   -     X1.g     X1.s     -
                        (>1.0)

chk1  tt    7.0   7.5   0.3      vgsl     X1.mn2   -     X1.g     X1.s     -
                        (<0.5)
```

chk2	tt2	2.0	8.5	-	cond_dcheck	X1.mn3	-	-	-	-
chk4	tt4	3.0	7.5	1.7 (>1.5)	cond_vgsu	X1.mn3	-	X1.g	X1.s	-
chk5	nmos1.1	71	72	1.8948 (>1.86)	vgsu	I19.I20.N00	-	I19.I20.A	I19.I20.0	-
chk5	nmos1.1	81	82	1.8968 (>1.86)	vgsu	I19.I20.N00	-	I19.I20.A	I19.I20.0	-
chk6	nmos1.1	71	72	1.8948 (>1.86)	vgsu	19.I20.N00	-	I19.I20.A	I19.I20.0	-
chk6	nmos1.1	81	82	1.8968 (>1.86)	vgsu	19.I20.N00	-	I19.I20.A	I19.I20.0	-
chk7	nmos1.1	31	32	1.8641 (>1.86)	vgsu	I19.I116.MN0	-	I19.I116.A	I19.I116.0	-
chk7	nmos1.1	83	84	1.8671 (>1.86)	vgsu	I19.I116.MN0	-	I19.I116.A	I19.I116.0	-
chk8	nmos1.1	31	32	1.8641 (>1.86)	vgsu	I19.I116.MN0	-	reset	0	-
chk8	nmos1.1	83	84	1.8671 (>1.86)	vgsu	I19.I116.MN0	-	reset	0	-

BJT Voltage Check

Spectre Syntax

```
dcheck title vbjt topnode=0|1 <model=[model1 model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vbcl=volt> <vbcu=volt> <vbel=volt> <vbeu=volt>
     <vbsl=volt> <vbsu=volt> <vcel=volt> <vceu=volt> <vcsl=volt> <vcsu=volt>
     <vesl=volt> <vesu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

SPICE Syntax

```
.dcheck title vbjt topnode=0|1 <model=[model1 model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vbcl=volt> <vbcu=volt> <vbel=volt> <vbeu=volt>
     <vbsl=volt> <vbsu=volt> <vcel=volt> <vceu=volt> <vcsl=volt> <vcsu=volt>
     <vesl=volt> <vesu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

Description

This command allows you to monitor bipolar junction transistor (BJT) voltages during a simulation run, and generates a report if the voltages exceed the specified upper and lower bounds, or meets the specified conditions. You can exclude a subset of the instances from the voltage check using the `xsubckt` or `xinst` arguments. If a threshold or condition is not specified for `dcheck` in the netlist file, a warning message is issued by the Virtuoso UltraSim simulator and `dcheck` is ignored during the simulation.

Arguments

`title`	Title of the voltage check.
`topnode=0│1`	Specifies whether the top-level node name or the hierarchical terminal name is to be reported in the dcheck report file. If set to 0, the hierarchical device terminal name is reported (default). If set to 1, the top-level node name is reported. If a top-level node name is not available, the hierarchical terminal name is used.
`model`	BJT voltage check is applied to transistors matching the model name (wildcards are supported).
`subckt`	BJT voltage check is applied to transistors belonging to all instances of the subcircuits listed (wildcards are supported).
`xsubckt`	BJT voltage check is excluded from instances of the subcircuits listed (wildcards are supported).
`inst`	BJT voltage check is applied to transistors belonging to the subcircuit instances listed (wildcards are supported).
`xinst`	BJT voltage check is excluded from the subcircuit instances listed (wildcards are supported). **Note:** The `inst` and `xinst` arguments can only be used to specify subcircuit instances, but not device instances.
`vbcl=`*`volt`*	Reports the condition if Vbc is less than the specified lower bound voltage value.
`vbcu=`*`volt`*	Reports the condition if Vbc is greater than the specified upper bound voltage value.

`vbel=`*`volt`*	Reports the condition if Vbe is less than the specified lower bound voltage value.
`vbeu=`*`volt`*	Reports the condition if Vbe is greater than the specified upper bound voltage value.
`vbsl=`*`volt`*	Reports the condition if Vbs is less than the specified lower bound voltage value.
`vbsu=`*`volt`*	Reports the condition if Vbs is greater than the specified upper bound voltage value.
`vcel=`*`volt`*	Reports the condition if Vce is less than the specified lower bound voltage value.
`vceu=`*`volt`*	Reports the condition if Vce is greater than the specified upper bound voltage value.
`vcsl=`*`volt`*	Reports the condition if Vcs is less than the specified lower bound voltage value.
`vcsu=`*`volt`*	Reports the condition if Vcs is greater than the specified upper bound voltage value.
`vesl=`*`volt`*	Reports the condition if Ves is less than the specified lower bound voltage value.
`vesu=`*`volt`*	Reports the condition if Ves is greater than the specified upper bound voltage. **Note:** The `vesu` argument and other arguments listed in this table can be used with constant parameters, but must be enclosed by single quotation marks (for example, `vesu='-par1'`).
`cond=`*`expression`*	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and variables: `vbc`, `vbe`, `vbs`, `vce`, `vcs`, `ves`, `l`, `w`. The expression can be a combination of linear and non-linear expressions. **Note:** The conditional check can be combined with the lower and upper threshold bounds mentioned in the Description.

`duration=`*`dtime`*	Reports the condition if device voltages are out of bounds for a duration of time longer than dtime (*`dtime`* default value is equal to the minimum time step of the simulation).
`time_window`	The time period specified for checking in which the first number is the start time point and the second number is the stop time point. For example, *`start1`* to *`stop1`* is the first time period and *`start2`* to *`stop2`* is the second time period. **Note:** Only ascending time points can be used (for example, *`start1`* < *`stop1`* < *`start2`* < *`stop2`*).
`probe=0|1`	Flag to probe node voltage for devices checked. If set to `0`, no probe is performed (default). If set to `1`, all node voltages for devices checked with `dcheck` are probed.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices

Example

Spectre Syntax:

```
dcheck chk1 vbjt model=[tt] vbeu=1.0 inst=[X1] xinst=[X1.X0] time_window=[5n 10u]
probe=1
dcheck chk2 vbjt vbeu=0.7 vbel=-0.5 inst=[i1]
dcheck chk3 vbjt vbeu=0.7 vbel=-0.5 inst=[i1] topnode=1
```

SPICE Syntax:

```
.dcheck chk1 vbjt model=[tt] vbeu=1.0 inst=[X1] xinst=[X1.X0] time_window=[5n 10u]
probe=1
.dcheck chk2 vbjt vbeu=0.7 vbel=-0.5 inst=[i1]
.dcheck chk3 vbjt vbeu=0.7 vbel=-0.5 inst=[i1] topnode=1
```

The command line of `chk1` checks voltages of all BJTs using the `tt` model in instance `X1` and its sub-hierarchy from transient time 5ns to 10us, excluding the `X1.X0` instance. BJTs that meet the vbeu>1 criteria are reported by the simulator. Where `probe=1`, all node voltages of the `tt` devices are probed.

The command line of `chk2` checks all BJT voltages in the instance `i1` and its sub-hierarchy. BJTs that meet the vbeu>0.7 or vbeu<-0.5 criteria are reported by the simulator.

The command line of `chk3` checks the same thing as `chk2` but reports the top-level node names rather than hierarchical terminal names in the dcheck report file.

Sample Output

Title	Model	From (ns)	To (ns)	v_curr (vth)	chk_type	Device	Collector	Base	Emitter	Substrate
chk1	tt	5.0	5.2	1.2 (>1.0)	vbeu	X1.q1	-	X1.b	X1.e	-
chk2	knpn	1	1.6	5.03608(>0.7)	vbeu	il.q0	-	il.net52	il.vss!	-
chk2	knpn	0	40	-0.57091(<-0.5)	vbe1	il.q2	-	il.iref	il.vdd!	-
chk3	knpn	1	1.6	5.03608(>0.7)	vbeu	il.q0	-	il.net52	vss!	-
chk3	knpn	0	40	-0.57091(<-0.5)	vbe1	il.q2	-	net35	vdd!	-

Resistor Voltage Check

Spectre Syntax

```
dcheck title vres topnode=0|1 <subckt=[subckt1 subckt2...]> <xsubckt=[xsubckt1
     xsubckt2...]> <inst=[inst1 inst2…]> <xinst=[xinst1 xinst2…]> <vpnl=volt>
     <vpnu=volt> <cond=expression> <duration=dtime> <time_window=[start1 stop1
     start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

SPICE Syntax

```
.dcheck title vres topnode=0|1 <subckt=[subckt1 subckt2...]> <xsubckt=[xsubckt1
     xsubckt2...]> <inst=[inst1 inst2…]> <xinst=[xinst1 xinst2…]> <vpnl=volt>
     <vpnu=volt> <cond=expression> <duration=dtime> <time_window=[start1 stop1
     start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

Description

This command allows you to monitor resistor voltages during a simulation run, and generates a report if the voltages exceed the specified upper and lower bounds, or meets the specified conditions. You can exclude a subset of the instances from the voltage check using the `xsubckt` or `xinst` arguments. If a threshold or condition is not specified for `dcheck` in the netlist file, a warning message is issued by the Virtuoso UltraSim simulator and `dcheck` is ignored during the simulation.

Arguments

`title`	Title of the voltage check.
`topnode=0|1`	Specifies whether the top-level node name or the hierarchical terminal name is to be reported in the dcheck report file. If set to 0, the hierarchical device terminal name is reported (default). If set to 1, the top-level node name is reported. If a top-level node name is not available, the hierarchical terminal name is used.
`subckt`	The voltage check is applied to resistors belonging to all instances of the subcircuits listed (wildcards are supported).
`xsubckt`	The voltage check is excluded from instances of the subcircuits listed (wildcards are supported).
`inst`	The voltage check is applied to resistors belonging to the subcircuit instances listed (wildcards are supported).
`xinst`	The voltage check is excluded from the subcircuit instances listed (wildcards are supported). **Note:** The `inst` and `xinst` arguments can only be used to specify subcircuit instances, but not device instances.
`vpnl=`*`volt`*	Reports the condition if Vpn is less than the specified lower bound voltage value.
`vpnu=`*`volt`*	Reports the condition if Vpn is greater than the specified upper bound voltage value. **Note:** The `vpnu` argument and other arguments listed in this table can be used with constant parameters, but must be enclosed by single quotation marks (for example, `vpnu='-par1'`).

`cond=`*`expression`*	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and the `vpn` variable. The expression can be a combination of linear and non-linear expressions. **Note:** The conditional check can be combined with the lower and upper threshold bounds mentioned in the Description.
`duration=`*`dtime`*	Reports the condition if device voltages are out of bounds for a duration of time longer than dtime (*`dtime`* default value is equal to the minimum time step of the simulation).
`time_window`	The time period specified for checking in which the first number is the start time point and the second number is the stop time point. For example, *`start1`* to *`stop1`* is the first time period and *`start2`* to *`stop2`* is the second time period. **Note:** Only ascending time points can be used (for example, *`start1`* < *`stop1`* < *`start2`* < *`stop2`*).
`probe=0|1`	Flag to probe node voltage for devices checked. If set to `0`, no probe is performed (default). If set to `1`, all node voltages for devices checked with `dcheck` are probed.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices **Note:** Set `preserve=all` if the specified resistor is subject to RC reduction.

Example

Spectre Syntax:

```
dcheck chk1 vres vpnu=1.0 inst=[X1] time_window=[5n 10u] probe=1
dcheck chk2 vres inst=[I19] xinst=[I19.I19.I3] vpnu=0.05
dcheck chk3 vres inst=[I19] xinst=[I19.I19.I3] vpnu=0.05 topnode=1
```

SPICE Syntax:

```
.dcheck chk1 vres vpnu=1.0 inst=[X1] time_window=[5n 10u] probe=1
.dcheck chk2 vres inst=[I19] xinst=[I19.I19.I3] vpnu=0.05
.dcheck chk3 vres inst=[I19] xinst=[I19.I19.I3] vpnu=0.05 topnode=1
```

The command line of `chk1` checks all resistors belonging to the `X1` instance and its sub-hierarchy from transient simulation time 5 ns to 10 us. The resistors that meet the vpnu>1.0 criteria are reported and the node voltages of all resistors inside `X1` are probed.

The command line of `chk2` checks all the resistors for instance `I19` and its sub-hierarchy, from which `I19.I19.I3` instance is excluded. The resistors that meet the vpnu>0.05 criteria are reported.

The command line of `chk3` checks the same thing as `chk2` but reports the top-level node names rather than hierarchical terminal names into dcheck report file.

Sample Output

Title	Model	From (ns)	To (ns)	v_curr(vth)	chk_type	Device	1	2
chk1	tt	5.0	5.2	1.2 (>1.0)	vpnu	X1.r1	X1.n1	X1.n2
chk2	R	30	31	0.061919(>0.05)	vpnu	I19.R2.r1	I19.R2.PLUS	I19.R2.MINUS
chk3	R	30	31	0.061919(>0.05)	vpnu	I19.R2.r1	I19.n5	I19.n6

Capacitor Voltage Check

Spectre Syntax

```
dcheck title vcap topnode=0|1 <subckt=[subckt1 subckt2...]> <xsubckt=[xsubckt1
     xsubckt2...]> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <vpnl=volt> <vpnu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

SPICE Syntax

```
.dcheck title vcap topnode=0|1 <subckt=[subckt1 subckt2...]> <xsubckt=[xsubckt1
     xsubckt2...]> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <vpnl=volt> <vpnu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1>
     <preserve=none|all>>
```

Description

This command allows you to monitor capacitor voltages during a simulation run, and generates a report if the voltages exceed the specified upper and lower bounds, or meet the specified conditions. You can exclude a subset of the instances from the voltage check using the `xsubckt` or `xinst` arguments. If a threshold or condition is not specified for `dcheck` in the netlist file, a warning message is issued by the Virtuoso UltraSim simulator and `dcheck` is ignored during the simulation.

Arguments

`title`	Title of the voltage check.
`topnode=0|1`	Specifies whether the top-level node name or the hierarchical terminal name is to be reported in the dcheck report file. If set to 0, the hierarchical device terminal name is reported (default). If set to 1, the top-level node name is reported. If a top-level node name is not available, the hierarchical terminal name is used.
`subckt`	The voltage check is applied to capacitors belonging to all instances of the subcircuits listed (wildcards are supported).
`xsubckt`	The voltage check is excluded from instances of the subcircuits listed (wildcards are supported).
`inst`	The voltage check is applied to capacitors belonging to the subcircuit instances listed (wildcards are supported).
`xinst`	The voltage check is excluded from the subcircuit instances listed (wildcards are supported). **Note:** The `inst` and `xinst` arguments can only be used to specify subcircuit instances, but not device instances.
`vpnl=`*`volt`*	Reports the condition if Vpn is less than the specified lower bound voltage value.
`vpnu=`*`volt`*	Reports the condition if Vpn is greater than the specified upper bound voltage value. **Note:** The `vpnu` argument and other arguments listed in this table can be used with constant parameters, but must be enclosed by single quotation marks (for example, `vpnu='-par1'`).

`cond=`*`expression`*	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and the `vpn` variable. The expression can be a combination of linear and non-linear expressions. **Note:** The conditional check can be combined with the lower and upper threshold bounds mentioned in the Description.
`duration=`*`dtime`*	Reports the condition if device voltages are out of bounds for a duration of time longer than dtime (*`dtime`* default value is equal to the minimum time step of the simulation).
`time_window`	The time period specified for checking in which the first number is the start time point and the second number is the stop time point. For example, *`start1`* to *`stop1`* is the first time period and *`start2`* to *`stop2`* is the second time period. **Note:** Only ascending time points can be used (for example, *`start1`* < *`stop1`* < *`start2`* < *`stop2`*).
`probe=0|1`	Flag to probe node voltage for devices checked. If set to `0`, no probe is performed (default). If set to `1`, all node voltages for devices checked with `dcheck` are probed.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices **Note:** Set `preserve=all` if the specified capacitor is subject to RC reduction.

Examples

Spectre Syntax:

```
dcheck chk1 vcap vpnu=1.0 inst=[X1] time_window=[5n 10u]
dcheck chk2 vcap xinst=[I19.I19.I3] vpnu=1.1
dcheck chk3 vcap vpnl=-5 preserve=all
dcheck chk4 vcap vpnl=-5 preserve=all topnode=1
```

SPICE Syntax:

```
.dcheck chk1 vcap vpnu=1.0 inst=[X1] time_window=[5n 10u]
.dcheck chk2 vcap xinst=[I19.I19.I3] vpnu=1.1
.dcheck chk3 vcap vpnl=-5 preserve=all
.dcheck chk4 vcap vpnl=-5 preserve=all topnode=1
```

The command line of `chk1` checks all the capacitors belonging to instance `X1` and its sub-hierarchy from transient time 5 ns to 10 us. The capacitors that meet the vpnu>1.0 criteria are reported.

The command line of `chk2` checks all the capacitors in the netlist file, excluding the `I19.I19.I3` instance and its sub-hierarchy. The capacitors that meet the vpnu>1.1 criteria are reported.

The command line of `chk3` checks all the capacitors in the netlist file. The capacitors that meet the vpnl<-5 criterion are reported.

The command line of `chk4` checks the same thing as `chk3` but reports the top-level node names rather than the hierarchical terminal names into the dcheck report file.

Sample Output

Title	Model	From (ns)	To (ns)	v_curr(vth)	chk_type	Device	1	2
chk1	tt	5.0	5.2	1.2 (>1.0)	vpnu	X1.c1	X1.n1	X1.n2
chk2	C	0	23.6	1.8827 (>1.1)	vpnu	C7	clk_p0_1x	0
chk3	C	0	40	-6.50024(<-5)	vpnl	il.c0	il.net6	il._net0
chk4	C	0	40	-6.50024(<-5)	vpnl	il.c0	il.net6	out

Diode Voltage Check

Spectre Syntax

```
dcheck title vdio topnode=0|1 <model=[model1 model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vpnl=volt> <vpnu=volt> <cond=expression>
     <duration=dtime> <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1>
     <preserve=none|all>
```

SPICE Syntax

```
.dcheck title vdio topnode=0|1 <model=[model1 model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vpnl=volt> <vpnu=volt> <cond=expression>
     <duration=dtime> <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1>
     <preserve=none|all>
```

Description

This command allows you to monitor diode voltages during a simulation run, and generates a report if the voltages exceed the specified upper and lower bounds, or meet the specified conditions. You can exclude a subset of the instances from the voltage check using the `xsubckt` or `xinst` arguments. If a threshold or condition is not specified for `dcheck` in the netlist file, a warning message is issued by the Virtuoso UltraSim simulator and `dcheck` is ignored during the simulation.

Arguments

`title`	Title of the voltage check.
`topnode=0|1`	Specifies whether the top-level node name or the hierarchical terminal name is to be reported in the dcheck report file. If set to 0, the hierarchical device terminal name is reported (default). If set to 1, the top-level node name is reported. If a top-level node name is not available, the hierarchical terminal name is used.
`model`	The diode voltage check is applied to resistors matching the model name (wildcards are supported).
`subckt`	The diode voltage check is applied to resistors belonging to all instances of the subcircuits listed (wildcards are supported).
`xsubckt`	The diode voltage check is excluded from instances of the subcircuits listed (wildcards are supported).
`inst`	The diode voltage check is applied to resistors belonging to the subcircuit instances listed (wildcards are supported).

`xinst`	The diode voltage check is excluded from the subcircuit instances listed (wildcards are supported). **Note:** The `inst` and `xinst` arguments can only be used to specify subcircuit instances, but not device instances.
`vpnl=`*`volt`*	Reports the condition if Vpn is less than the specified lower bound voltage value.
`vpnu=`*`volt`*	Reports the condition if Vpn is greater than the specified upper bound voltage value. **Note:** The `vpnu` argument and other arguments listed in this table can be used with constant parameters, but must be enclosed by single quotation marks (for example, `vpnu='-par1'`).
`cond=`*`expression`*	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and the `vpn` variable. The expression can be a combination of linear and non-linear expressions. **Note:** The conditional check can be combined with the lower and upper threshold bounds mentioned in the Description.
`duration=`*`dtime`*	Reports the condition if device voltages are out of bounds for a duration of time longer than dtime (*`dtime`* default value is equal to the minimum time step of the simulation).
`time_window`	The time period specified for checking in which the first number is the start time point and the second number is the stop time point. For example, *`start1`* to *`stop1`* is the first time period and *`start2`* to *`stop2`* is the second time period. **Note:** Only ascending time points can be used (for example, *`start1`* < *`stop1`* < *`start2`* < *`stop2`*).
`probe=0|1`	Flag to probe node voltage for devices checked. If set to `0`, no probe is performed (default). If set to `1`, all node voltages for devices checked with `dcheck` are probed.

`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices

Examples

Spectre Syntax:

```
dcheck diochk1 vdio vpnu=2 vpnl=0
dcheck diochk2 vdio subckt=[pll] xinst=[I19.I19.I3] vpnl=0
dcheck diochk3 vdio subckt=[pll] xinst=[I19.I19.I3] vpnl=0 topnode=1
```

SPICE Syntax:

```
.dcheck diochk1 vdio vpnu=2 vpnl=0
.dcheck diochk2 vdio subckt=[pll] xinst=[I19.I19.I3] vpnl=0
.dcheck diochk3 vdio subckt=[pll] xinst=[I19.I19.I3] vpnl=0 topnode=1
```

The command line of `diochk1` checks all the diodes in the netlist file. The diodes that meet the vpnu>2 or vpnl<0 criteria are reported by the simulator.

The command line of `diochk2` checks the diodes in the instances of subcircuit `p11` and its sub-hierarchy, excluding the `I19.I19.I3` instance. The diodes that meet the vpnl<0 criterion are reported by the simulator.

The command line of `diochk3` checks the same thing as `diochk2` but reports the top-level node names rather than the hierarchical terminal names into the dcheck report file.

Sample Output

Title	Model	From (ns)	To (ns)	v_curr(vth)	chk_type	Device	1	2
diochk1	D	29	30	-0.5 (< 0)	vpnl	d1	n1	n2
diochk1	D	5	25	4.5 (> 2)	vpnu	d1	n1	n2
diochk2	D	0	85	-1.0499 (<0)	vpnl	I19.D2	I19.vss	I19.vcom
diochk2	D	0	85	-1.26648 (<0)	vpnl	I19.D3	I19. vss	I19.vcop
diochk3	D	0	85	-1.0499 (<0)	vpnl	I19.D2	0	vcom
diochk3	D	0	85	-1.26648 (<0)	vpnl	I19.D3	0	vcop

JFET/MESFET Voltage Check

Spectre Syntax

```
dcheck title vjft topnode=0|1 <model=[model1, model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vgdl=volt> <vgdu=volt> <vdsl=volt> <vdsu=volt>
     <vdbl=volt> <vdbu=volt> <vgsl=volt> <vgsu=volt> <vgbl=volt> <vgbu=volt>
     <vsbl=volt> <vsbu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

SPICE Syntax

```
.dcheck title vjft topnode=0|1 <model=[model1, model2...]> <subckt=[subckt1
     subckt2...]> <xsubckt=[xsubckt1 xsubckt2...]> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <vgdl=volt> <vgdu=volt> <vdsl=volt> <vdsu=volt>
     <vdbl=volt> <vdbu=volt> <vgsl=volt> <vgsu=volt> <vgbl=volt> <vgbu=volt>
     <vsbl=volt> <vsbu=volt> <cond=expression> <duration=dtime>
     <time_window=[start1 stop1 start2 stop2 ...]> <probe=0|1> <preserve=none|all>
```

Description

This command allows you to monitor junction field effect transistor (JFET) or metal semiconductor field effect transistor (MESFET) voltages during a simulation run, and generates a report if the voltages exceed the specified upper and lower bounds, or meets the specified conditions. You can exclude a subset of the instances from the voltage check using the `xsubckt` or `xinst` arguments. If a threshold or condition is not specified for `dcheck` in the netlist file, a warning message is issued by the Virtuoso UltraSim simulator and `dcheck` is ignored during the simulation.

Arguments

`title`	Title of the voltage check.
`topnode=0|1`	Specifies whether the top-level node name or the hierarchical terminal name is to be reported in the dcheck report file. If set to 0, the hierarchical device terminal name is reported (default). If set to 1, the top-level node name is reported. If a top-level node name is not available, the hierarchical terminal name is used.
`model`	MOS voltage check is applied to transistors matching the model name (wildcards are supported).

`subckt`	MOS voltage check is applied to transistors belonging to all instances of the subcircuits listed (wildcards are supported).
`xsubckt`	MOS voltage check is excluded from instances of the subcircuits listed (wildcards are supported).
`inst`	MOS voltage check is applied to transistors belonging to the subcircuit instances listed (wildcards are supported).
`xinst`	MOS voltage check is excluded from the subcircuit instances listed (wildcards are supported).
	Note: The `inst` and `xinst` arguments can only be used to specify subcircuit instances, but not device instances.
`vgdl=`*`volt`*	Reports the condition if Vgd is less than the specified lower bound voltage value.
`vgdu=`*`volt`*	Reports the condition if Vgd is greater than the specified upper bound voltage value.
`vdsl=`*`volt`*	Reports the condition if Vds is less than the specified lower bound voltage value.
`vdsu=`*`volt`*	Reports the condition if Vds is greater than the specified upper bound voltage value.
`vdbl=`*`volt`*	Reports the condition if Vdb is less than the specified lower bound voltage value.
`vdbu=`*`volt`*	Reports the condition if Vdb is greater than the specified upper bound voltage value.
`vgsl=`*`volt`*	Reports the condition if Vgs is less than the specified lower bound voltage value.
`vgsu=`*`volt`*	Reports the condition if Vgs is greater than the specified upper bound voltage value.
`vgbl=`*`volt`*	Reports the condition if Vgb is less than the specified lower bound voltage value.
`vgbu=`*`volt`*	Reports the condition if Vgb is greater than the specified upper bound voltage value.
`vsbl=`*`volt`*	Reports the condition if Vsb is less than the specified lower bound voltage value.

`vsbu=`*`volt`*	Reports the condition if Vsb is greater than the specified upper bound voltage. **Note:** The `vsbu` argument and other arguments listed in this table can be used with constant parameters, but must be enclosed by single quotation marks (for example, `vsbu='-par1'`).
`cond=`*`expression`*	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and variables: `vgd`, `vds`, `vdb`, `vgs`, `vgb`, `vsb`, `l`, `w`. The expression can be a combination of linear and non-linear expressions. **Note:** The conditional check can be combined with the lower and upper threshold bounds mentioned in the Description.
`duration=`*`dtime`*	Reports the condition if device voltages are out of bounds for a duration of time longer than dtime (*`dtime`* default value is equal to the minimum time step of the simulation).
`time_window`	The time period specified for checking in which the first number is the start time point and the second number is the stop time point. For example, *`start1`* to *`stop1`* is the first time period and *`start2`* to *`stop2`* is the second time period. **Note:** Only ascending time points can be used (for example, *`start1`* < *`stop1`* < *`start2`* < *`stop2`*).
`probe=0|1`	Flag to probe node voltage for devices checked. If set to `0`, no probe is performed (default). If set to `1`, all node voltages for devices checked with `dcheck` are probed.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices

Examples

Spectre Syntax:

```
dcheck chk1 vjft model=[tt] inst=[X1] xsubckt=[Reg*] vgsu=1.0 vgsl=0.5 probe=1
dcheck chk2 vjft model=[tt2] cond=((vgs<-3 || vds>3) && l<0.2u)
```

SPICE Syntax

```
.dcheck chk1 vjft model=[tt] inst=[X1] xsubckt=[Reg*] vgsu=1.0 vgsl=0.5 probe=1
.dcheck chk2 vjft model=[tt2] cond='(vgs<-3 || vds>3) && l<0.2u'
```

The command line of `chk1` in the netlist file tells the Virtuoso UltraSim simulator to check all JFET model devices using model `tt` in block `X1` and its sub-hierarchy. Instances of subcircuits with names that match `*Reg` are excluded (if instances of the `Reg*` subcircuits are not part of the `X1` instance, their sub-hierarchies are also excluded). The devices that meet the vgs>1 or vgs<0.5 criteria are reported by the simulator. Where `probe=1`, all node voltages of the `tt` devices are probed.

The command line of `chk2` tells the simulator to check all JFET model devices using model `tt2`, whether vgs<-3 or vds>3, and when the JFET length is less than 0.2 um. If the conditions are met, the devices are reported by the simulator.

Dynamic Power Checking

The section introduces the Virtuoso UltraSim simulator dynamic power analyses. These commands allow you to perform a power analysis using probe and measure statements, and report the power consumed by each element and subcircuit in the design.

.measure/power

Description

The power measure statement monitors the average, maximum, minimum, peak-to-peak, RMS, and integral (total energy) of the instantaneous power consumed by the elements or subcircuit. If the netlist filename is `circuit.sp`, the value files are called `circuit.meas#` and `circuit.mt#`.

Examples

In the following example

```
.measure tran power_max max `v(xtop.x23.out) * x0(xtop.x23.out)` from=0ns to=1us
```

tells the Virtuoso UltraSim simulator to measure the maximum power of port out of instance `xtop.x23`, excluding all other lower hierarchical subcircuit ports.

The next example

```
.measure tran power_min min `v(xtop.x23.out) * x(xtop.x23.out)` from=0ns to=1us
```

tells the simulator to measure the maximum power of port out of instance `xtop.x23` and all instances below it.

The next example

```
.measure tran power_avg avg `v(1) * i1(r1)` from=0ns to=1us
```

tells the simulator to measure the average power on element `r1` in the circuit.

The next example

```
.measure tran energy integ ` v(xtop.x23.out) * x(xtop.x23.out)` from=0ns to=10us
```

tells the simulator to measure the integral power (total energy) of port `out` of instance `xtop.x23` and all instances below it.

.probe/power

Description

The power probe statement is used to set up power probes on elements or subcircuits for a specified output quantity. Two output files are created for this probe statement. If the netlist filename is `circuit.sp`, the output files are called `circuit.expr.trn` and `circuit.expr.dsn`.

Examples

In the following example

```
.probe tran power=par(`v(xtop.x23.out) * x0(xtop.x23.out)`)
```

tells the Virtuoso UltraSim simulator to probe the power of port `out` of instance `xtop.x23`, excluding all other lower hierarchical subcircuit ports.

The next example

```
.probe tran power=par(`v(xtop.x23.out) * x(xtop.x23.out)`)
```

tells the simulator to probe the power of port `out` of instance `xtop.x23` and all instances below it.

The next example

```
.probe tran power=par(`v(1) * i1(r1)`)
```

tells the simulator to probe the power on element `r1` in the circuit.

Node Activity Analysis

Spectre Syntax

```
usim_nact title node=[node1 node2...] <limit=value> <start=time> <stop=time>
     <type=max_vo|avg_vo|...> <sort=inc|dec> <param=[max_vo ...] <swingvth=value>
```

SPICE Syntax

```
.usim_nact title node=[node1 node2 ...] <limit=value> <start=time> <stop=time>
     <type=max_vo|avg_vo|...> <sort=inc|dec> <param=[max_vo ...] <swingvth=value>
```

Description

This command sets up the node activity analysis for the specified nodes. The analysis reports the following parameters for each node:

- Maximum and average voltage overshoot (VO)
- Maximum and average voltage undershoot (VU)
- Maximum, average, and minimum rise times
- Maximum, average, and minimum fall times
- Signal probability of being high and low
- Capacitance
- Number of toggles
- Full-swing or non-full-swing status

A time window can be specified for the analysis performed. If a wildcard (`*`) is used in the node names, the number of nodes for which data is printed can be limited using the `limit` keyword. For more information about wildcards, see "Wildcard Rules" on page 55.

The output data is printed to a file with the extension `.nact`. For example, if the name of the input netlist file is `circuit.sp`, then the output file is named `circuit.nact`. For multiple node activity analysis commands, all activity reports are saved in the `.nact` file in the same order as the commands were issued.

The number of nodes for which data is printed in the output file can be limited. This is to restrict the size of the output file if the circuit is large. If the limit is not specified, then data for all the nodes is printed to the file.

The nodes can be sorted before being printed to the file. Each of the column names in the output file can be treated as a sort variable. That is, it can be used for sorting, and only one column can be used for sorting. The sorting order, ascending or descending, can also be specified. By default, the nodes are sorted in increasing order of their names (that is, in alphabetical order). If a sort variable is specified, then it is used for sorting. For example, if `type=max_vo sort=inc` is specified in the command card, the nodes are sorted in increasing order of their maximum VO value. If many nodes have the exact same maximum VO, then they are sorted according to the default sorting criterion, by increasing order of their names.

By default, the command reports all parameters for each node. The number of reported parameters can be limited using the `param` statement.

Arguments

`title`	Title of the node activity analysis.
`[node1 node2...]`	Specifies the nodes that need to be checked; accepts wildcards (`*`).
`limit`	Limits the number of nodes which are output to the file to *n*. The *n* nodes that rank highest, according to the specified criterion, are printed to the file.
`start`	Start time of the check window. If not specified, the default is 0.
`stop`	Stop time of the check window. If not specified, the default is the stop time of the simulation.
`type`	Sets the column name to be sorted.
	Note: You can use only one column name for sorting.
`sort` =(`inc` | `dec`)	Sets the sorting order:
	`inc`, sorts in increasing order of the column values.
	`dec`, sorts in decreasing order of the column values.
`param`	Defines the column names printed in the report. The column names that are not listed are not printed. If the `param` keyword is not specified, all the column names are printed.

`swingvth`	Defines the voltage threshold for detecting nodes that are not full-swing. A node is not considered to be full-swing if: ■ high-level voltage cannot reach the `vdd-swingvth` value. or ■ low-level voltage cannot be under the `gnd+swingvth` value.

The reported column names, specified in the `.usim_nact` file, are described below:

Column Name Descriptions

`max_vo`	Maximum voltage overshoot (VO) at the node during time window (reference level is the high level defined by `.usim_opt vdd` or the highest available DC voltage level - see log file for `vdd` value)
`t_max_vo`	Time when maximum VO occurs
`avg_vo`	Average VO at the node during time window (reference level is vdd)
`max_vu`	Maximum voltage undershoot (VU) at the node during time window (reference level is 0V)
`t_max_vu`	Time when maximum VU occurs
`avg_vu`	Average VU at the node during time window (reference level is 0V)
`max_rise`	Maximum rise time at the node during time window, measured from vl to vh (use `.usim_opt vl/vh` to define threshold) **Note:** Use `max_rt` in netlist file.
`t_max_rise`	Time when maximum rise time occurs
`min_rise`	Minimum rise time at the node during time window, measured from `vl` to `vh` (use `.usim_opt vl/vh` to define threshold) **Note:** Use `min_rt` in netlist file.
`t_min_rise`	Time when minimum rise time occurs
`avg_rise`	Average rise time at the node during time window, measured from `vl` to `vh` (use `.usim_opt vl/vh` to define threshold) **Note:** Use `avg_rt` in netlist file.

`max_fall`	Maximum fall time at the node during time window, measured from `vh` to `vl` (use `.usim_opt vl/vh` to define threshold) **Note:** Use `max_ft` in netlist file.
`t_max_fall`	Time when maximum fall time occurs
`min_fall`	Minimum fall time at the node during time window, measured from `vh` to `vl` (use `.usim_opt vl/vh` to define threshold) **Note:** Use `min_ft` in netlist file.
`t_min_fall`	Time when minimum fall time occurs
`avg_fall`	Average fall time at the node during time window, measured from `vh` to `vl` (use `.usim_opt vl/vh` to define threshold) **Note:** Use `avg_ft` in netlist file.
`probe_h`	Percentage of transient simulation time node was in logic 1 state (above `vh`)
`probe_l`	Percentage of transient simulation time node was in logic `0` state (below `vl`)
`cap`	Total average node capacitance including device capacitances
`toggle`	Number of times node toggled from low to high or high to low (high level defined by `vh` and low level defined by `vl`)
`half_swing`	Indicates whether a node is full-swing. A value of `1` indicates a non-full-swing node, and a value of `0` indicates a full-swing node.

Examples

Spectre Syntax:

```
usim_nact example limit=10 type=max_vo sort=inc
```

SPICE Syntax:

```
.usim_nact example limit=10 type=max_vo sort=inc
```

tells the Virtuoso UltraSim simulator to display the top 10 nodes which have the highest VO.

VO is the difference between the node and supply voltage, when the node voltage is greater than the supply voltage. If the node voltage is less than the supply voltage, VO is assumed to be 0.

VU is defined as the difference between the ground and node voltage. If the node voltage is higher than the ground level, VU is assumed to be 0. Rise time is the time it takes the node to go from 30% to 70% of the difference between the supply voltage and ground voltage. Fall time is the time it takes the node to go from 70% to 30% of the difference between the supply voltage and ground voltage.

Spectre Syntax:

```
usim_nact example1 type=cap sort=dec param=[cap toggle max_rt]
```

SPICE Syntax:

```
.usim_nact example1 type=cap sort=dec param=[cap toggle max_rt]
```

tells the simulator to create a report with all nodes ordered after their node capacitance and prints the node capacitance, maximum rise time, time at which maximum rise time appears, and number of toggles.

Spectre Syntax:

```
usim_nact check_swing node=[out1 out2 in] param=[toggle half_swing] swingvth=0.1
start=10ns stop=40ns
```

SPICE Syntax:

```
.usim_nact check_swing node=[out1 out2 in] param=[toggle half_swing] swingvth=0.1
start=10ns stop=40ns
```

tells the simulator to check whether the three nodes `out1`, `out2` and, `in` are full-swing based on the specified swing threshold value (`swingvth=0.1`), and prints the `half_swing` flag together with the number of toggles in the report file.

Node Glitch Analysis

Spectre Syntax

```
usim_nact title analysis=glitch node=[node1 node2...] <vurelth=value>
     <vuabsth=value> <vurelrecth=value> <vuabsrecth=value> <vorelth=value>
     <voabsth=value> <vorelrecth=value> <voabsrecth=value>
     <type=max_vo|avg_vo|max_vu|avg_vu> <sort=inc|dec> <start=time> <stop=time>
     <limit=value> <numlevel=value>
```

SPICE Syntax

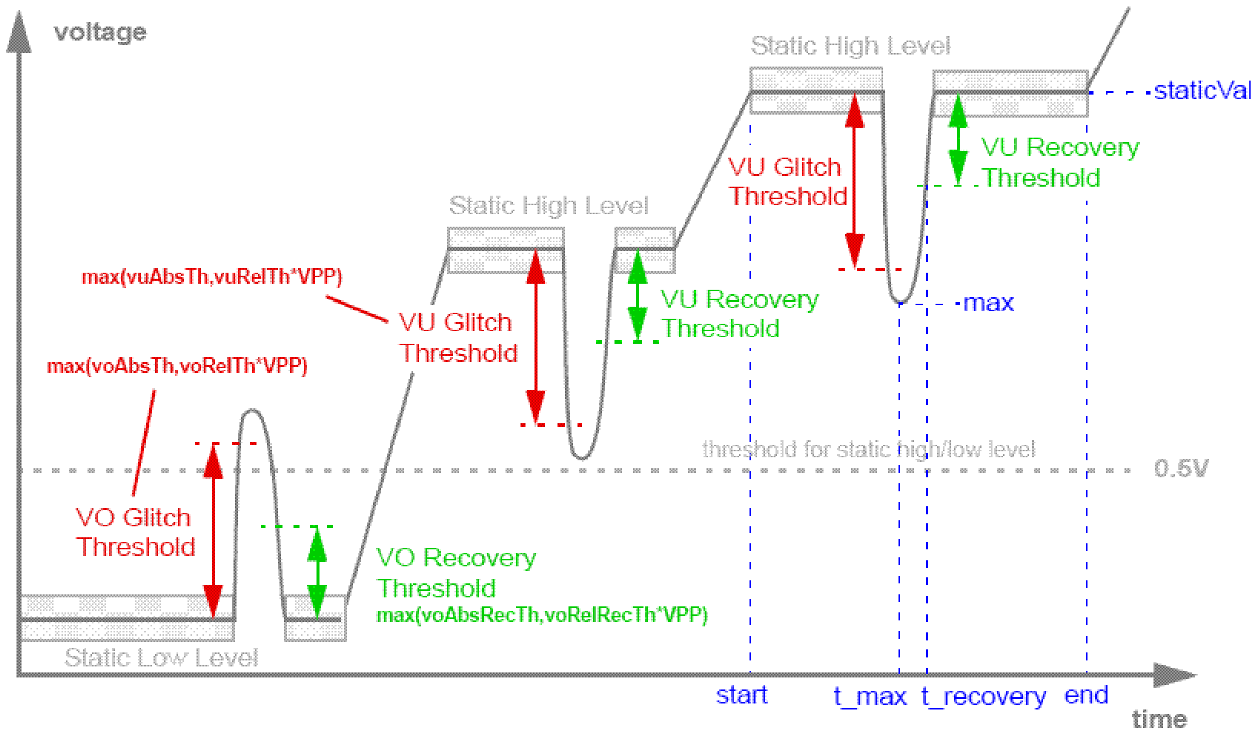
```
.usim_nact title analysis=glitch node=[node1 node2...] <vurelth=value>
     <vuabsth=value> <vurelrecth=value> <vuabsrecth=value> <vorelth=value>
     <voabsth=value> <vorelrecth=value> <voabsrecth=value>
     <type=max_vo|avg_vo|max_vu|avg_vu> <sort=inc|dec> <start=time> <stop=time>
     <limit=value> <numlevel=value>
```

Description

This command sets up the node glitch analysis for the specified nodes. Node glitch analysis is a post-processing feature, which detects glitches in reference to static voltage levels of a signal.

Node glitch analysis is performed as follows:

1. Static voltage levels (where the voltage level is constant) of all signals defined as levels are determined.
2. All static levels below or equal to 0.5V are considered as static low level.
3. All static levels above 0.5V are considered as static high level.
4. Glitches are detected in reference to the static voltage levels. The software detects overshoot glitches in reference to static low level and undershoot glitches in reference to static high level.



The report contains one line for all glitches occurring during one static level. If one signal has multiple static levels and each static level contains glitches, one line is reported for each static level. The following parameters are reported for the glitches of each static voltage level:

- `avg`: Specifies the average glitch voltage level, that is, the average of maximum value of all glitches within one static level.
- `max`: Specifies the maximum glitch voltage level, that is, the voltage level of the maximum glitch within the static level.
- `t_max`: Specifies the time of the maximum glitch.
- `t_recovery`: Specifies the time taken by the signal to recover from the glitch.
- `staticVal`: Specifies the static voltage level.
- `Vpp`: Specifies the high-level voltage used to calculate glitch threshold based on relative tolerances.
- `start`: Specifies the start time of the static voltage level.

- `end`: Specifies the end time of the static voltage level.

The report can be sorted based on overshoot glitches, undershoot glitches, average glitch voltage level, and maximum glitch voltage level. In addition, the values can be arranged in increasing or decreasing order.

Overshoot glitches can be identified in the report by the reported static low level (below/equal to 0.5V). Undershoot glitches can be identified in the report by the reported static high level.

Arguments

`title`	Title of the node glitch analysis.
`[node1 node2...]`	Specifies the nodes that need to be checked; accepts wildcards (`*`).
`vurelth`	Specifies the relative tolerance for undershoot glitch detection. *Default*: `0.1`
`vuabsth`	Specifies the absolute tolerance for undershoot glitch detection. *Default*: `0.5V`
`vurelrecth`	Specifies the relative tolerance for undershoot glitch recovery. *Default*: `0.1`
`vuabsrecth`	Specifies the absolute tolerance for undershoot glitch recovery. *Default*: `0.5V`
`vorelth`	Specifies the relative tolerance for overshoot glitch detection. Default: `0.1`
`voabsth`	Specifies the absolute tolerance for overshoot glitch detection. *Default*: 0.5V
`vorelrecth`	Specifies the relative tolerance for overshoot glitch recovery. *Default*: `0.1`
`voabsrecth`	Specifies the absolute tolerance for overshoot glitch recovery. *Default*: `0.5V`

`type`	Specifies the criteria of sorting, which could be one of the following: ■ `max_vo`: Sorts the results based on maximum overshoot or undershoot value. ■ `avg_vo`: Sorts the results based on average overshoot or undershoot glitch value. ■ `max_vu`: Sorts the results based on maximum overshoot or undershoot value. ■ `avg_vu`: Sorts the results based on average overshoot or undershoot glitch value. **Note:** The report does not differentiate between overshoot and undershoot glitches. Therefore, you will see the same sorting results when you use `max_vo` or `max_vu`. Similarly, you will see the same sorting results when you use `avg_vo` or `avg_vu`.
`sort =(inc | dec)`	Sets the order for sorting: `inc`: Sorts in increasing order of the column values. `dec`: Sorts in decreasing order of the column values.
`start`	Specifies the start time of the check window. *Default*: `0`
`stop`	Specifies the stop time of the check window. If not specified, the default is the stop time of the simulation.
`limit`	Limits the number of nodes, which are output to the file. When this option is specified, the software prints the glitch information for only the specified number of nodes. The nodes that rank higher based on the specified criteria are printed first. *Default*: unlimited
`numlevel`	Limits the number of static levels per signal. When this option is specified, the software prints the glitch information for only the specified number of static levels per signal. *Default*: `5`

Examples

Spectre Syntax:

```
usim_nact glitch analysis=glitch type=max_vo sort=inc node=[ vdd1 vdd2 vss1 vss2
out1 out2 ] vurelth=0.1 vuabsth=0.25 vurelrecth=0.02 vuabsrecth=0.05 vorelth=0.1
voabsth=0.25 vorelrecth=0.02 voabsrecth=0.05
```

SPICE Syntax:

```
.usim_nact glitch analysis=glitch type=max_vo sort=inc node=[ vdd1 vdd2 vss1 vss2
out1 out2 ] vurelth=0.1 vuabsth=0.25 vurelrecth=0.02 vuabsrecth=0.05 vorelth=0.1
voabsth=0.25 vorelrecth=0.02 voabsrecth=0.05
```

tells the Virtuoso UltraSim simulator to perform a glitch analysis on the nodes `vdd1`, `vdd2`, `vss1`, `vss2`, `out1`, and `out2` using the defined threshold values for glitch detection, and recovery. The report is sorted based on the maximum overshoot glitches (in increasing order).

Power Analysis

Spectre Syntax

```
usim_pa title subckt inst=[inst1 inst2 ...] port=[porta portb ...] <depth=level>
     <sort=max|avg|rms> <subckt_limit=n1> <power=[on|off]> <time_window=[start1
     stop1 start2 stop2 ...]> <fast_mode=0|1>
```

SPICE Syntax

```
.usim_pa title subckt inst=[inst1 inst2 ...] port=[porta portb ...] <depth=level>
     <sort=max|avg|rms> <subckt_limit=n1> <power=[on|off]> <time_window=[start1
     stop1 start2 stop2 ...]> <fast_mode=0|1>
```

Description

This command is used to set up a power analysis on specified subcircuits. It reports the average, maximum, and RMS current at the ports of subcircuits, child subcircuits, and grandchild subcircuits for a specified level of hierarchy (report output is a text file). Included in the report is the time point at which the maximum value is reached. If there are more than two time points with the same maximum value, the first occurrence is reported.

Optionally, the command can be used to report the average, maximum, and RMS power consumed by the subcircuit and its subcircuits within the specified hierarchical level.

Note: The total (generated and consumed) power at the top level is not reported in the power analysis.

The current and power information is also output to a text file. The file name convention is *netlistname.pa* and the file contains three sections:

- Current information for the ports (first section)
- Power information for the ports (second section)
- Subcircuit information (third section)

If the circuit is simulated more than once (for example, when using alter or sweep), the file name convention changes to *netlistname.runnumber.pa*.

Note: The report can be imported into Microsoft® Excel for additional analyses.

Arguments

`inst`	List of instances to be checked. If not specified, all subcircuits at the hierarchical level are analyzed. Wild card is supported.
`port`	Specifies the subcircuit port to be checked (port names in the subcircuit definition, not the ports in the instances). The Virtuoso UltraSim simulator reports the current (power optional) information for specified ports. If the specified ports are ports for the child subcircuit, the simulator reports the port information at the child subcircuit level. If the ports are also ports for the grandchild subcircuit, the simulator reports the information at the grandchild subcircuit level. This reporting structure continues until the specified hierarchical level is reached. If not specified, all ports at the specified hierarchical level are automatically reported. **Note:** When a port is specified, the Virtuoso UltraSim simulator does not report the subcircuit power consumption (that is, the *Subckt Power Summary* section is omitted from the output file).
`depth`	The hierarchical depth of the subcircuits to be checked (default is 1).
`sort=max|avg|rms`	Sorts the report by the specified value, in decreasing order. The values include: ■ `avg`: The average instantaneous power for specified time intervals ■ `max`: The maximum instantaneous power during the entire simulation ■ `rms`: The RMS of the instantaneous power for specified time intervals. If there is more than one sorting criterion, the first one is used and the second one ignored. For example: If `sort=avg` in the first `usim_pa` subcircuit and `sort=max` in the second `usim_pa` subcircuit, only `sort=avg` is used.
`subckt_limit=n1`	Limits the number of subcircuits to be reported (default is `infinity`).

`power`	Turns specified power value `on` or `off` (default is `off`). The values include: ■ `power=off`: Only current information is reported ■ `power=on`: Current and power information is reported ■ `start` and `stop`: Time window for check (must be paired) If `start` and `stop` are not specified, `start` = 0 s and `stop` = end of the simulation.
`fast_mode`	Specifies whether or not to check MOS gates. The values include: `fast_mode=0`: Checks all detected ports (default) `fast_mode=1`: Skips ports that are MOS gates

pa_elemlen

The default length for subcircuit instance names is 20 characters. Use the `pa_elemlen` option to change the name length. For example, `usim_opt pa_elemlen=64` sets the maximum name length to 64 characters.

Example 1

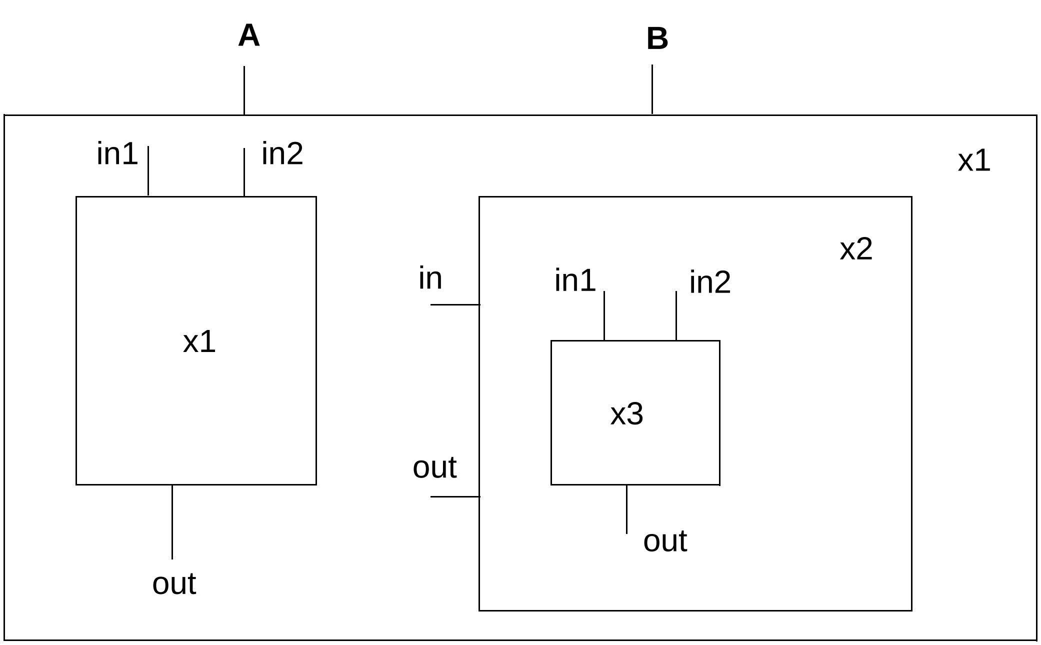
The report format is determined by the sorting criteria. For example, block `x1` has two ports, `A` and `B` (`in`/`out` in subcircuit definition), with blocks `x1.x1` and `x1.x2` (see Figure 8-1 on page 422).

Sample netlist file:

```
x1 A B sub_x1
.subckt sub_x1 in out
   .
   .
   .
    x1 in1 in2 out sub_x1_x1
    x2 in out sub_x1_x2
    .subckt sub_x1_x1 in1 in2 out
      .
      .
      .
```

```
    .ends sub_x1_x1
    .subckt sub_x1_x2 in out
      .
      .
      .
        x3 in1 in2 out sub_x1_x2_x3
        .subckt sub_x1_x2_x3 a b c
         .
         .
         .
        .ends sub_x1_x2_x3
    .ends sub_x1_x2
   .
   .
   .
.ends sub_x1
```

Figure 8-1 Power Analysis Report Format Example



The remaining blocks and ports are arranged in the following order:

- Block `x1.x1` has three ports: `in1`, `in2`, and `out` (`in1`, `in2`, and `out` in subcircuit definition)
- Block `x1.x2` has two ports: `xin` and `xout` (`in` and `out` in subcircuit definition)

- Block `x1.x2` contains block `x1.x2.x3`
- Block `x1.x2.x3` has three ports: `in1`, `in2`, and `out` (`a`, `b`, and `c` in subcircuit definition)

Example 2

Spectre Syntax:

```
usim_pa example2 subckt inst=[x1] depth=3 sort=max power=off time_window=[10n 50n]
```

SPICE Syntax:

```
.usim_pa example2 subckt inst=[x1] depth=3 sort=max power=off time_window=[10n 50n]
```

The first section of the output file includes the following information:

```
Time: from 10n to 50n
*** Port Current Summary *******************

               Max(A)     Avg (A)    RMS(A)     Max Time
x1.B           600e-3     600e-3     500e-3     15e-9
x1.x2.x3.b     550e-3     300e-3     300e-3     12e-9
x1.A           540e-3     300e-3     600e-3     15e-9
x1.x1.in1      530e-3     400e-3     500e-3     15e-9
x1.x1.out      520e-3     300e-3     100e-3     10e-9
x1.x2.in       500e-3     300e-3     300e-3     25e-9
x1.x1.in2      500e-3     200e-3     1e-3       30e-9
x1.x2.out      400e-3     400e-3     200e-3     50e-9
x1.x2.x3.c     350e-3     500e-3     200e-3     30e-9
x1.x2.x3.a     200e-3     100e-3     100e-3     20e-9
```

Example 3

```
usim_pa example3 subckt inst=[x1] depth=3 sort=max power=on time_window=[10n 50n]
```

The first section of the output file *netlistname.pa* is the same as in Example 2. The second and third sections of the file include the following information:

```
*** Port Power Summary **********

               Max(w)     Avg (w)    RMS (w)    Max time
```

```
x1.B           40e-3     50e-3     60e-3     25e-9
x1.A           35e-3     30e-3     20e-3     30e-9
x1.x1.in2      32e-3     40e-3     5e-3      40e-9
x1.x2.x3.b     30e-3     10e-3     7e-3      45e-9
x1.x1.in1      30e-3     20e-3     10e-3     30e-9
x1.x1.out      29e-3     4e-3      2e-3      35e-9
x1.x2.out      23e-3     20e-3     30e-3     40e-9
x1.x2.in       10e-3     20e-3     40e-3     50e-9
x1.x2.x3.a      6e-3     15e-3     8e-3      13e-9
x1.x2.x3.c     4e-3      3e-3      6e-3      16e-9

*** Subckt Power Summary ******

               Max(w)    Avg (w)   RMS (w)   Max time
x1             60e-3     50e-3     0e-3      10e-9
x1.x1          30e-3     30e-3     10e-3     20e-9
x1.x2          30e-3     20e-3     10e-3     30e-9
x1.x2.x3       10e-3     20e-3     10e-3     35e-9
```

Example 4

```
usim_pa example4 subckt inst=[x1.x2] port=[in] depth=1 sort=max power=on
time_window=[10n 50n]
```

Since the port is specified, the Virtuoso UltraSim simulator does not report the power consumption for the subcircuit (output file only has two sections: *Port Current Summary* and *Port Power Summary*).

```
Time: from 10n to 50n
*** Port Current Summary *******************

               Max(A)    Avg (A)   RMS(A)    Max Time
 x1.x2.in      500e-3    300e-3    300e-3    25e-9
```

*** Port Power Summary ******

	Max(w)	Avg (w)	RMS (w)	Max time
x1.x2.in	30e-3	20e-3	10e-3	30e-9

Example 5

```
usim_pa example5 subckt inst=[x*] port=[in*] depth=3 sort=avg power=off
time_window=[1n 2n]
```

tells the Virtuoso UltraSim simulator to print out the current consumption for all ports that have names starting with `in` and for all subcircuits that have names starting with `x`. The hierarchical depth is limited to 3, the report is sorted by the `avg` value, and the time window is from 1 ns to 2 ns.

Example 6

```
usim_pa example6 subckt depth=3 sort=max power=on time_window=[100p 2n]
```

tells the simulator to print out current and power consumption for all ports, and power consumption for all subcircuits within a hierarchical depth of 3 for time window 100 ps to 2 ns. The report is sorted by the `max` value.

Wasted and Capacitive Current Analysis

Spectre Syntax

```
usim_pa title currents inst=[inst1 inst2 ...] [static=on|off] time_window=[start1
     stop1 start2 stop2 ...]
```

SPICE Syntax

```
.usim_pa title currents inst=[inst1 inst2 ...] [static=on|off] time_window=[start1
     stop1 start2 stop2 ...]
```

Description

Capacitive current is the current charging or discharging of a capacitance node. Wasted current is the current flowing between two voltage sources that does not contribute to any switching functions. There are two types of wasted current: Static and dynamic. Static wasted current is the portion of wasted current flowing in circuits that are not switching. Dynamic wasted current is the portion of wasted current flowing in circuits which are actively switching.

The `usim_pa currents` command is used to analyze the capacitive, and static and dynamic wasted currents for specified circuits. The analysis results report the RMS and average values of currents consumed by the subcircuit, and its subcircuits within the specified hierarchical level. The current information is output to a `netlistname.pa` text file.

The wasted and capacitive current check applies only to digital type of designs including SRAMs and other memories. This feature does not apply to analog designs.

Arguments

`title`	Title for the current analysis.
`inst1, inst2`	List of subcircuit instances to be analyzed. If instances are not specified, the entire circuit is analyzed. **Note:** Wildcards (`*`) are supported.
`static=on|off`	Static and dynamic wasted current is reported if `static=on` (default is `static=off` and only the total wasted current is reported).
`time_window`	The time period for checking

Example 1

In the following Spectre syntax example

```
usim_pa example1 currents inst=x1 static=on start=100n stop=1000n
```

In the first example, the Virtuoso UltraSim simulator reports the capacitive current, as well as the static and dynamic wasted currents for instance `x1` over the simulation window of time=100 ns to time=1000 ns.

The following report is generated:

```
.TITLE 'This file is :./mult16_vec.pa'
Time: from 100n to 1000n
*** Subckt Current Summary ***


 x1
 Average capacitive current:     6.181e+03 uA
 RMS capacitive current:         2.632e+04 uA
 Average wasted current:         1.861e+02 uA
 RMS wasted current:             3.077e+03 uA
 Average static wasted current:  2.446e+00 uA
 RMS static wasted current:      2.446e+00 uA
 Average dynamic wasted current: 1.887e+02 uA
```

Example 2

In the following SPICE syntax example

```
.usim_pa example2 currents static=on
```

In the second example, the simulator reports the capacitive current, and static and dynamic wasted currents of the whole circuit over the entire simulation window.

Power Checking

Over Current (Excessive Current) Check

Spectre Syntax

```
pcheck title exi elem=[elem1 <elem2>...] <ith=current_threshold>
     <tth=time_duration> <time_window=[start1 stop1 start2 stop2 ...]>
     <preserve=none|all>
```

SPICE Syntax

```
.pcheck title exi elem=[elem1 <elem2>...] <ith=current_threshold>
     <tth=time_duration> <time_window=[start1 stop1 start2 stop2 ...]>
     <preserve=none|all>
```

Description

Based on the specified element list (current threshold, over current duration time, and checking windows), the Virtuoso UltraSim simulator reports in a `.pcheck` file which elements over a specific time period have current over the threshold for a time period equal to or greater than the specified duration. If no window is specified, the entire simulation period is used.

Arguments

`title`	User defined title name for check.
`exi`	Keyword for over current check.
`elem1 <elem2…>`	List of element instance names to be checked.

`ith`	Defines current threshold (default is 10 uA).
`tth`	Defines duration time (default is 5 ns).
`time_window`	Defines the checking window.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves active devices only ■ **all** preserves all devices or nodes, including passive devices

Examples

Spectre Syntax:

```
pcheck check1 exi elem=[XIO.M12 XIO.M32] ith=5e-3 tth=10n
pcheck check2 exi elem=[X1.X132.*] ith=1e-4 tth=10n time_window=[0 1u 3u 10u]
pcheck check3 exi elem=[*] ith=2e-3 tth=100n
```

SPICE Syntax:

```
.pcheck check1 exi elem=[XIO.M12 XIO.M32] ith=5e-3 tth=10n
.pcheck check2 exi elem=[X1.X132.*] ith=1e-4 tth=10n time_window=[0 1u 3u 10u]
.pcheck check3 exi elem=[*] ith=2e-3 tth=100n
```

Note: The element instance list can only contain element names or be enclosed by `I( )`, single quotation marks `' '`, or double quotation marks `" "` [if only a wildcard `*` is used, it requires `I( )` or quotation marks]. For more information about wildcards, see "Wildcard Rules" on page 55.

Over Voltage (Excessive Node Voltage) Check

Spectre Syntax

```
pcheck title exv node=[node1 <node2...>] <vmin=value> <vmax=value>
     <tth=time_duration> <time_window=[start1 stop1 start2 stop2 ...]>
     <preserve=none|all> <option=0|1>
```

SPICE Syntax

```
.pcheck title exv node=[node1 <node2...>] <vmin=value> <vmax=value>
     <tth=time_duration> <time_window=[start1 stop1 start2 stop2 ...]>
     <preserve=none|all> <option=0|1>
```

Description

Based on the specified node list, voltage threshold, over voltage duration time, and checking windows, the Virtuoso UltraSim simulator reports in a `.pcheck` file which nodes over a specific time period have voltage over the threshold for a time period equal to or greater than the specified duration. If no window is specified, the entire simulation period is used.

Arguments

`title`	User defined title name for check.
`exv`	Keyword for over voltage check.
`node1 <node2…>`	Names of nodes to be checked (wildcards are supported).
`vmin=`*`value`*	Defines minimum voltage level. If not defined, `vmin` checking is not performed by simulator.
`vmax=`*`value`*	Defines maximum voltage level. If not defined, `vmax` checking is not performed by simulator.
`tth=`*`time_duration`*	Defines duration time (default is 5 ns).
`time_window`	Defines the checking window.
`preserve=none|all`	Defines whether all devices are preserved. ■ **none** preserves nodes after RC reduction ■ **all** preserves all nodes, including nodes removed during RC reduction
`option=0|1`	Defines which voltage threshold is used to report the nodes. ■ **0** reports any of the specified nodes with voltages above `vmax` or below `vmin` for a duration time longer than `tth` (default) ■ **1** reports any of the specified nodes if its voltage falls between `vmin` and `vmax` for a duration time longer than `tth`

Examples

Spectre Syntax:

```
pcheck exv1 exv node=[*] vmax=0.8 tth=1n
pcheck exv2 exv node=[*] vmin=0
```

```
pcheck exv3 exv node=[*] vmin=0 vmax=0.8 option=0 tth=1n
pcheck exv4 exv node=[*] vmin=0.35 vmax=0.75 option=1 tth=1n
pcheck exv5 exv node=[*] vmin=0.35 vmax=0.75 option=1 tth=1n preserve=all
pcheck exv6 exv node=[*] vmin=0.35 vmax=0.75 option=1 tth=1n preserve=all
time_window=[8n 12n 18n 22n]
```

SPICE Syntax:

```
.pcheck exv1 exv node=[*] vmax=0.8 tth=1n
.pcheck exv2 exv node=[*] vmin=0
.pcheck exv3 exv node=[*] vmin=0 vmax=0.8 option=0 tth=1n
.pcheck exv4 exv node=[*] vmin=0.35 vmax=0.75 option=1 tth=1n
.pcheck exv5 exv node=[*] vmin=0.35 vmax=0.75 option=1 tth=1n preserve=all
.pcheck exv6 exv node=[*] vmin=0.35 vmax=0.75 option=1 tth=1n preserve=all
time_window=[8n 12n 18n 22n]
```

DC Path Leakage Current Check

Spectre Syntax

```
pcheck title dcpath <ith=threshold_current> <tth=time_duration> <node=[node1
     node2...]> <inst=[inst1 inst2]> <xinst=[xinst1 xinst2]>
     <period=period_time|delay=delay_time> <time_window=[start1 stop1 start2
     stop2 ...]>

pcheck title dcpath <ith=threshold_current> <tth=time_duration> <node=[node1,
     node2...]> <inst=[inst1 inst2]> <xinst=[xinst1 xinst2]> <at=[time1 time2...]>
```

SPICE Syntax

```
.pcheck title dcpath <ith=threshold_current> <tth=time_duration> <node=[node1
     node2...]> <inst=[inst1 inst2]> <xinst=[xinst1 xinst2]>
     <period=period_time|delay=delay_time> <time_window=[start1 stop1 start2
     stop2 ...]>

.pcheck title dcpath <ith=threshold_current> <tth=time_duration> <node=[node1,
     node2...]> <inst=[inst1 inst2]> <xinst=[xinst1 xinst2]> <at=[time1 time2...]>
```

Description

The Virtuoso UltraSim simulator reports the DC conducting paths between specified voltage source nodes. All reported DC conducting paths are written into a file and it has an `.pcheck` extension. To qualify as a conducting path, each segment in the path must, at a minimum, carry the threshold current specified by the parameter `ith`.

A voltage source node is a node which is directly connected to a voltage source (includes DC, PWL, and PULSE voltage sources). The ground node is also a voltage source node. Nodes connected to current sources, HDL/Verilog-A/C models, and drivers defined in VEC or VCD files are not qualified. If nodes are not specified, the simulator checks for DC paths between all voltage sources. If only one node is specified, the DC path between the node and ground is reported. If a time point or frame is not specified, the entire simulation period is checked. Currents and voltages reported in the DC path report correspond to values at the beginning of the measured time window.

Arguments

`title`	User defined title name for check.
`dcpath`	Keyword for DC path check.
`ith`	Threshold current (default value is 50 uA).
`tth`	Duration time (default is 5 ns).
`nodelist`	List of voltage source nodes to be checked.
`inst`	Specifies subcircuit (instance) to be checked by simulator. If not specified, the entire circuit is checked. Wildcard characters can be used with `subckt`. For more information about wildcards, see “Wildcard Rules” on page 55.
`xinst`	Specifies subcircuit (instance) to be excluded from check. Wildcard characters can be used with this argument.
`period`	Specifies that the DC current path is checked for every period, starting from the beginning of each time frame as defined by `start` and `stop`.
`delay`	Specifies that the DC current path is checked at each time defined by *t+delay_time*. `t` designates the time an input stimulus change occurs. If both `period` and `delay` are not specified, the DC path is checked in the time frame defined by `start` and `stop`. **Note:** `period` and `delay` cannot be used simultaneously.
`at`	Specifies that the DC current path is checked at the time defined by *at=time1*.
`time_window`	Specifies time frame for checking (default is full transient simulation).

Examples

In the following Spectre example

```
pcheck dc1 dcpath ith=1e-6 tth=10n node=[vdd gnd] delay=5n time_window=[10n 210n]
```

tells the Virtuoso UltraSim simulator to check the DC current path between `vdd` and `gnd` after any input stimulus change, with a delay of 5 ns. The DC current path is checked during the 10 ns and 210 ns time frame. The DC current path is reported in the `netlist.pcheck` file if the DC current path exceeds 1 uA and lasts longer than 10 ns.

In the following SPICE example

```
.pcheck dc2 dcpath ith=1e-6 node=[vddh vddl] period=10n time_window=[10n 210n]
```

tells the simulator to check the DC current path between `vddh` and `vddl` every 10 ns, starting at 10 ns and stopping at 210 ns. The DC current path is reported if the DC current path exceeds 1 uA.

In the following Spectre example

```
pcheck dc3 dcpath node=[vcc vss] at=[130n 150n]
```

tells the simulator to check the DC current path between `vcc` and `vss` at 130 ns and 150 ns. The DC current path is reported if the DC current path exceeds the default value of 50 uA when checked.

The next example

```
pcheck dc4 dcpath inst=[IDIGITAL] xinst=[IDIGIAL.IOSC] ith=10u tth=10n
```

tells the simulator to check the DC current path between any two voltage sources over the entire simulation time. The DC current path is reported if the DC current path exceeds 10 uA and last longer than 10 ns. Only the `IDIGIAL` block is checked (the `IOSC` block inside `IDIGIAL` is excluded).

High Impedance Node Check

Spectre Syntax

```
pcheck title zstate node=[node1 <node2...>] <fanout=0|1|2> <xsubckt=[xsubckt1
     xsubckt2 ...]> <psubckt=[psubckt1 psubckt2 ...]> <ztime=ztime>
     <time_window=[start1 stop1 start2 stop2 ...]>
```

SPICE Syntax

```
.pcheck title zstate node=[node1 <node2...>] <fanout=0|1|2> <xsubckt=[xsubckt1
     xsubckt2 ...]> <psubckt=[psubckt1 psubckt2 ...]> <ztime=ztime>
     <time_window=[start1 stop1 start2 stop2 ...]>
```

Description

Based on the specified node name list (high-z duration time and checking windows), the Virtuoso UltraSim simulator reports in a `.pcheck` file which nodes over what time period were in high-z state for a time period equal to or greater than the specified duration. If no window is specified, the whole simulation period is used.

Along with reporting the node name, the high-z state check reports the times when the high z-state begins and ends, as well as the time error (time error is defined as the time the high-z state exceeds the specified time).

A node is considered to be in high-z state if there is only a high impedance or no connection from the node to a voltage source or ground. The following conditions in the path can produce a high-z state:

- MOSFET is switched off (Vgs<Vth and Ids < Ith)

  Note: See the <u>Notes</u> section for the definition of lth.

- JFET is switched off (Vgs<Vpinchoff)
- Resistor bigger than Rth

  Note: See the <u>Notes</u> section for the definition of Rth.

- BJT considered off if Vbe<=0.4 and Ic<=50 nA
- Diode considered to be off for V<0.6 V
- Verilog-A module with no channel connection (capacitor, diode, mutual inductor, and current source with <1 pA)

Arguments

`title`	User defined title name for check.
`zstate`	Keyword for high-impedance node check.
`node1 <node2…>`	List of node names to be checked.

`fanout=0|1|2`	Optional connection option. If `fanout=0`, all listed nodes are checked (default). If `fanout=1`, only those nodes connected to the metal oxide semiconductor field-effect transistor (MOSFET) gate are checked. If `fanout=2`, only those nodes connected to the bulk or body of the MOSFET are checked (default is 0).
`xsubckt`	Defines subcircuit cells that are excluded from the check when '*' is used in the node name list (wildcard `*` is supported). For more information about wildcards, see "Wildcard Rules" on page 55.
`psubckt`	Checks high-z state for the I/O ports of the specified subcircuit. This is applied only when '*' is specified.
`ztime`	Defines duration time in high-z state (default is 5 ns).
`time_window`	Defines the time period for checking.

Examples

Spectre Syntax:

```
pcheck z_check1 zstate node=[xram.*] fanout=1 ztime=50n

pcheck z_check2 zstate node=[*] ztime=1.0e-8 time_window=[1u 9u]
xsubckt=[inv1* ?and]
```

SPICE Syntax:

```
.pcheck z_check1 zstate node=[xram.*] fanout=1 ztime=50n

.pcheck z_check2 zstate node=[*] ztime=1.0e-8 time_window=[1u 9u]
xsubckt=[inv1* ?and]
```

Notes

- The node instance list can only contain node names and must be enclosed by `v( )`, single quotation marks `' '`, or double quotation marks `" "` [if only a wildcard `*` is used, it requires `v( )` or quotation marks]. For more information about wildcards, see "Wildcard Rules" on page 55.
- When a wildcard is used, the expanded node instance list does not include nodes located within RC networks. You should always review the Virtuoso UltraSim simulator log file for all reported floating nodes (use the `warning_limit_float` option to print floating nodes).
- The MOSFET threshold current Ith can be changed by using the following option:

```
.usim_opt pck_mos_ids = value
```

The default value of Ith is 10nA.

- The resistance threshold value Rth can be changed by using the following option:

```
.usim_opt res_open = value
```

The default value of Rth is 100 Mohm.

Hot Spot Node Current Check

Spectre Syntax

```
pcheck title hotspot node=[node1 <node2 ...>] <ratio=ratio> <fanout=0|1|2>
     <xsubckt=[xsubckt1 xsubckt2 ...]> <psubckt=[psubckt1 psubckt2 ...]>
     <time_window=[start1 stop1 start2 stop2 ...]>
```

SPICE Syntax

```
.pcheck title hotspot node=[node1 <node2 ...>] <ratio=ratio> <fanout=0|1|2>
     <xsubckt=[xsubckt1 xsubckt2 ...]> <psubckt=[psubckt1 psubckt2 ...]>
     <time_window=[start1 stop1 start2 stop2 ...]>
```

Description

The Virtuoso UltraSim simulator reports the average charging and discharging current statistics for specified nodes during a checking window (the statistics are output to a `.hotspot` report). If a checking window is not specified, the entire simulation period is used.

Only the nodes, for which the sum of the charging and discharging average current is larger than the hot spot factor (default is 0.5) multiplied by the sum of the node with the largest current, are reported.

Arguments

`title`	User-defined title name for check.
`hotspot`	Keyword for hot spot check.
`node1 <node2…>`	List of node names to be checked.
`ratio`	Defines the hot spot factor (0 <= ratio <= 1; default is 0.5).

`fanout=0|1|2`	Optional connection option. If `fanout=0`, all listed nodes are checked (default). If `fanout=1`, only those nodes connected to the metal oxide semiconductor field-effect transistor (MOSFET) gate are checked. If `fanout=2`, only those nodes connected to the bulk or body of the MOSFET are checked (default is 0).
`xsubckt`	Defines subcircuit cells that are excluded from the check when '*' is used in the node name list.
`psubckt`	Checks hot spot for I/O ports of specified subcircuit. Only applied when '*' is specified.
`time_window`	Defines the time period for checking.

Examples

In the following Spectre example

```
pcheck hot_chk1 hotspot node=[xtop.x1.*]
```

tells the Virtuoso UltraSim simulator to report the average current statistics for all nodes in the `xtop.x1` block.

In the following SPICE example

```
.pcheck hot_chk2 hotspot node=[*] ratio=0.8 time_window=[1u 9u]
```

tells the simulator to report the average current statistics for all nodes. Since the hot spot factor is 0.8, only the nodes with a sum of charging and discharging current larger than 80% of maximum current are reported.

Notes

- The node instance list can only contain node names and must be enclosed by either `v()`, single quotation marks (`''`), or double quotation marks (`""`). If only a wildcard (`*`) is used, the node names need to use `v()` or quotation marks. For more information about wildcards, see "Wildcard Rules" on page 55.
- Nodes connected to voltage or ground sources are excluded from the hot spot node check.
- If multiple start and stop pairs are used in one statement, the hot spot node check is performed for each time window. If multiple hot spot statements are included in a netlist file, each statement must have an unique title (nodes in different statements are checked separately).
- Internal nodes within RC networks are currently excluded from the hot spot node check.

Sample Output

Title	node_name	Icin(uA)	Icout(uA)	from(ns)	to(ns)
hotspot_chk1	vpp	1548.47	1645.73	0	500
hotspot_chk1	x5.n2n1486	1574.65	1284.41	0	500
hotspot_chk1	x5.n2n1422	1484.39	1371.63	0	500
hotspot_chk1	x5.n2n1485	1495.16	1311.18	0	500
hotspot_chk1	x5.n2n1484	1329.68	1306.82	0	500
hotspot_chk1	x4.n1n646	1022.95	1028.15	0	500
hotspot_chk1	x5.n2n1488	1134.72	869.648	0	500
hotspot_chk1	x3.n1n646	903.956	909.934	0	500
hotspot_chk1	x5.nc	842.582	879.417	0	500
hotspot_chk1	Total Current	74885.4	74489.4	0	500

In this sample report output:

- `Icin` is the average charging current flowing into the capacitances connected to the node
- `Icout` is the average discharging current flowing out of the capacitances connected to the node
- The checking window duration is listed in the `from(ns)` and `to(ns)` columns
- The hot spot factor ratio = 0.5

  The node with the largest current is `vpp`, which has a sum of charging and discharging average current of 3194.2 uA. Multiplied by the ratio 0.5, the sum of charging and discharging average current is 1597.1 uA. Based on this ratio, all nodes with a current sum larger than 1597.1 uA are included in the report.

  Note: If the hot spot factor ratio is changed to 0.6, the `x5.nc` and `x3.n1n646` are excluded from the report.

Floating Gate Induced Leakage Current Check

Spectre Syntax

```
pcheck title floatdcpath <ith=threshold_current> <node=[node1, node2...]>
     <inst=[inst1 inst2]> <xinst=[xinst1 xinst2]> time_window=[time1 time2...]
     period=time5 at=[time6 time7...]
```

SPICE Syntax

```
.pcheck title floatdcpath <ith=threshold_current> <node=[node1, node2...]>
     <inst=[inst1 inst2]> <xinst=[xinst1 xinst2]> time_window=[time1 time2...]
     period=time5 at=[time6 time7...]
```

Description

The Virtuoso UltraSim simulator detects Hi-Z nodes and forces their associated fanout transistors to be turned `on`. If the operation forms any conducting paths between voltage source nodes through the transistor with leakage current larger than the threshold value, then these paths are reported in a `.pcheck` file. To qualify as a conducting path, each segment in the path must carry the threshold current specified by the `ith` parameter.

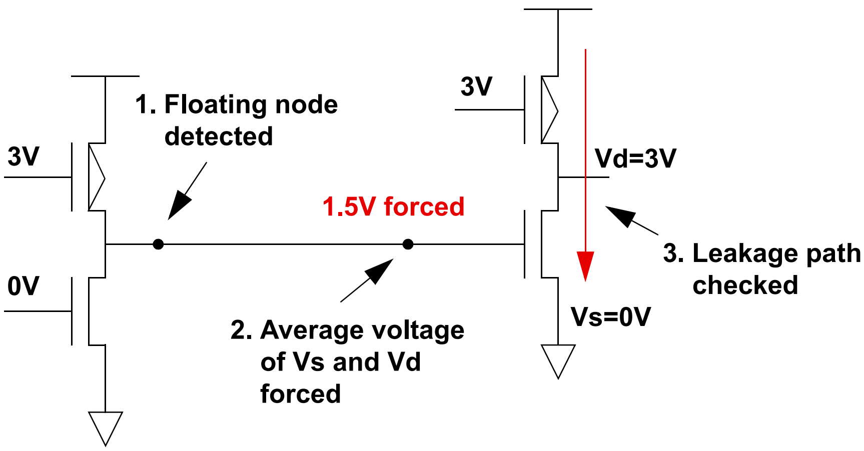
Note: The definitions of Hi-Z nodes are described in the High Impedance Node Check section.

A voltage source node is a node which is directly connected to a voltage source (includes DC, PWL, and PULSE voltage sources). The ground node is also a voltage source node.

Note: Nodes connected to current sources, HDL/Verilog-A/C models, and drivers defined in VEC or VCD files do not qualify.

If nodes are not specified, the simulator checks for DC paths between all voltage sources. If only one node is specified, the DC path between the node and ground is reported.

Figure 8-2 Floating Gate Induced Leakage Current Check Overview



Arguments

`title`	User-defined title name for check.
`floatdcpath`	Keyword for floating gate induced leakage current check.
`ith`	Threshold current (default value is 10 uA).
`node`	List of voltage source nodes to be checked. Wildcards are supported.
`inst`	Specifies subcircuit instance to be checked by simulator. If not specified, the entire circuit is checked. Wildcard characters can be used with the subcircuit.
`xinst`	Specifies subcircuit instances to be excluded from the check. Wildcard characters can be used with this argument.
`time_window`	Time window in which checking is performed.
`period`	Check is performed at every time period, starting at the beginning of `time_window` (default value is 10 ns or 1% of transient time, whichever time value is longer). Minimum period allowed is 1 ns.
`at`	Specifies time point for checking (ignored if `time_window` is specified).

Examples

In the following example

```
.pcheck dc2 floatdcpath node=[vcc vss] ith=50u at=[130n 150n]
```

tells the Virtuoso UltraSim simulator to check the DC current path between `vcc` and `vss` at 130 ns and 150 ns. The DC current path is reported if the path exceeds 50 uA during the check.

In the next example

```
.pcheck dc1 floatdcpath time_window=[200n 600n 1200n 1600n] period=[100n]
```

tells the simulator to check the DC current path between all source nodes at 100 ns intervals between 200 ns and 600ns, and 1200 ns and 1600 ns.

Excessive Rise and Fall Time Check (EXRF)

Spectre Syntax

```
pcheck title exrf node=[node1 <node2...>]  <fanout=0|1|2>  <rise=rise_time>
     <fall=fall_time> <utime=u_value> <vlth=logic_low_voltage>
     <vhth=logic_high_voltage> <time_window=[start1 stop1 start2 stop2 ...]>
     <separate_file=0|1>
```

SPICE Syntax

```
.pcheck title exrf node=[node1 <node2...>]  <fanout=0|1|2>  <rise=rise_time>
     <fall=fall_time> <utime=u_value> <vlth=logic_low_voltage>
     <vhth=logic_high_voltage> <time_window=[start1 stop1 start2 stop2 ...]>
     <separate_file=0|1>
```

Description

The Virtuoso UltraSim simulator reports (in a `.pcheck` file) the nodes that have excessive rise or fall time over a specific time period based on the specified list of nodes, logic voltage thresholds, and time checking windows.

If no time checking window is specified, the entire simulation period is used.

Arguments

`title`	User-defined title name for check.
`exrf`	Keyword for excessive rise/fall time check.
`node1 <node2…>`	List of node names to be checked.
`fanout=0|1|2`	Optional connection option. If `fanout=0`, all listed nodes are checked (default). If `fanout=1`, only those nodes connected to the metal oxide semiconductor field-effect transistor (MOSFET) gate are checked. If `fanout=2`, only those nodes connected to the bulk or body of the MOSFET are checked (default is 0).
`rise`	Transition time from logic low voltage to logic high voltage. The default value is 5 ns.
`fall`	Transition time from logic high voltage to logic low voltage. The default value is 5 ns.
`utime`	Time duration of the node stays between the logic low and logic high volltage without making a transition. The default value is 5 ns.
`vith`	Logic low threshold voltage. The default value os 0.3Vdd.
`vhth`	Logic low threshold voltage. The default value os 0.7Vdd.
`time_window`	Defines the time period for checking.

Example

```
.pcheck  exrf node = [ x1.x2.*]  fanout=1 rise=6n  fall=4n  vlth=0.4 vhth=2.6
+ time_window = [100n 2000n]
```

This command checks if the signal voltage values at the nodes `x1.x2.*` have excessive rise and fall times between 100 ns and 2000 ns. A violation is reported in the `.pcheck` file if the signal rise time exceeds 6 ns, or the signal fall time exceeds 4 ns, or the U-state time exceeds the default value of 5 ns.

Timing Analysis

The Virtuoso UltraSim simulator allows you to perform timing analysis on specific nodes through a set of commands starting with `usim_ta`. These commands should be directly embedded in the netlist file, or in a separate file that is included in the netlist file using the `include` command. The timing check errors are reported in the `.ta` file. If the netlist file is `circuit.sp`, then the `.ta` file is named circuit `.ta`.

Timing check statements can be embedded within a subcircuit definition. In this case, they apply only to the nodes local to the host circuit, and their check titles are appended by the circuit calls from the top level in the circuit hierarchy. Timing check statements also support the parameters `depth` = *`value`* and `subckt` = *`name`* simulation output statements (see "Supported SPICE Format Simulation Output Statements" on page 130 for more information). Nodes analyzed with `usim_ta` are automatically saved as waveforms.

For example,

Spectre Syntax:

```
usim_ta example setup node=n1 edge=rise ref_node=clk ref_edge=rise setup_time=2n
subckt=INV depth=2
```

SPICE Syntax:

```
.usim_ta example setup node=n1 edge=rise ref_node=clk ref_edge=rise setup_time=2n
subckt=INV depth=2
```

tells the Virtuoso UltraSim simulator to report the setup timing errors for all nodes that match `n*` in the subcircuit `INV` and one level below in the circuit hierarchy. See the following sections for timing check statements descriptions.

The timing analysis checks supported by the Virtuoso UltraSim simulator include:

- Hold Check on page 444
- Pulse Width Check on page 446
- Setup Check on page 448
- Timing Edge Check on page 451

Hold Check

Spectre Syntax

```
usim_ta title hold node=node1 edge=rise|fall|both ref_node=node2
     ref_edge=rise|fall|both hold_time=time <window=window_size>
     <vl=logic_0_threshold> <vh=logic_1_threshold>
     <vrl=logic_0_threshold> <vrh=logic_1_threshold> <depth=value>
     <subckt=name> <start=time1> <stop=time2>
```

SPICE Syntax

```
.usim_ta title hold node=node1 edge=rise|fall|both ref_node=node2
     ref_edge=rise|fall|both hold_time=time <window=window_size>
     <vl=logic_0_threshold> <vh=logic_1_threshold>
     <vrl=logic_0_threshold> <vrh=logic_1_threshold> <depth=value>
     <subckt=name> <start=time1> <stop=time2>
```

Description

This command is used to report hold timing errors on the specified nodes with respect to a reference node. A hold timing error occurs when a permissible signal transition occurs between the times `t_ref` and `t_ref+hold_time` (if `hold_time>0`), or between `t_ref+hold_time` and `t_ref+window_size` (if `hold_time<0`). Here, `t_ref` is the time point when a permissible reference transition occurs.

A permissible transition occurs when the waveform crosses the corresponding logic threshold. For example, when a waveform crosses the logic 1 threshold while rising, it has a RISE transition. If the logic 0 state and logic 1 state thresholds for the signal, reference, or both, are not specified on the command card, then the default values are used. The default values can be set using the command `usim_opt` `vl` = *value* `vh` = *value.*

Arguments

`title`	The title of the current timing analysis.
`node`	Specifies the node on which the hold timing check is performed. Wildcards are supported in the node name (see Chapter 3, "Simulation Options").

`edge=rise|fall|both`	The permissible transition type for the signal node. `rise`, a low-to-high transition is the permissible transition. `fall`, a high-to-low transition is the permissible transition. `both`, a low-to-high or a high-to-low transition is a permissible transition.
`ref_node`	The name of the reference node. Only a single node is allowed.
`ref_edge=rise|fall|both`	The permissible transition for the reference node. `rise`, a low-to-high transition is the permissible transition. `fall`, a high-to-low transition is the permissible transition. `both`, a low-to-high or a high-to-low transition is a permissible transition.
`hold_time`	The hold time. It can be positive or negative. If negative, the `window` parameter must be specified.
`window=`*`window_size`*	The time window after the reference transition. This parameter must be specified when the hold time is negative.
`vl=`*`logic_0_threshold`*	The threshold of logic 0 state for a signal. If the signal has a value less than `vl`, it is considered to be logic 0.
`vh=`*`logic_1_threshold`*	The threshold of logic 1 state for a signal. If the signal has a value greater than `vh`, it is considered to be logic 1.
`vrl=`*`logic_0_threshold`*	The threshold of logic 0 state for a reference. If the reference has a value less than `vrl`, it is considered to be logic 0.
`vrh=`*`logic_01threshold`*	The threshold of logic 1 state for a reference. If the reference has a value greater than `vrh`, it is considered to be logic 1.

`depth=`*`value`*	Specifies the depth in the circuit hierarchy that a wildcard name applies. If it is set as one, only the nodes at the current level is applied (default value is infinity).
`subckt=`*`name`*	Specifies the subcircuit that this statement applies in. By default, it applies in the top level. If the statement is already in a subcircuit definition, this parameter is ignored. Setting this parameter is equivalent to defining the statement within a subcircuit declaration.
`start=time1`	Timing analysis start time. The Virtuoso UltraSim simulator does not perform a timing analysis for simulation `time` < `time1`.
`stop=time2`	Timing analysis end time. The simulator does not perform a timing analysis for simulation `time` > `time2`.

Examples

In the following Spectre example

```
usim_ta ta_all hold example1 node=n1 edge=rise ref_node=clk ref_edge=fall
hold_time=2n
```

tells the Virtuoso UltraSim simulator to report hold timing errors if the `rise` transitions on node `n1` occur within the 2 ns, after a `fall` transition on the node `clk`.

In the following SPICE example

```
.usim_ta ta_all hold example2 node=n2 edge=both ref_node=clk ref_edge=fall
hold_time=-1n window=5n
```

tells the simulator to report hold timing errors if the `rise` or `fall` transitions on node `n2` occur within the time interval of 1 ns before the `fall` transition of the node `clk`, and 5 ns after the `clk fall` transition.

Pulse Width Check

Spectre Syntax

```
usim_ta title pulsew node=node1 tmin_low=min_low_time tmax_low=max_low_time
     tmin_high=min_high_time tmax_high=max_high_time <vl=logic_0_threshold>
     <vh=logic_1_threshold> <depth=value> <subckt=name> <start=time1>
     <stop=time2>
```

SPICE Syntax

```
.usim_ta title pulsew node=node1 tmin_low=min_low_time tmax_low=max_low_time
     tmin_high=min_high_time tmax_high=max_high_time <vl=logic_0_threshold>
     <vh=logic_1_threshold> <depth=value> <subckt=name> <start=time1>
     <stop=time2>
```

Description

This command is used to report pulse width errors on the waveforms of the specified nodes. Pulse width is defined to be the time interval during which the signal in a node stays in the low or high state. A pulse width error occurs when the pulse width of a signal falls outside the range (`min_low_time`, `max_low_time`) for the logic 0 state, or the range (`min_high_time`, `max_high_time`) for the logic 1 state.

If the logic 0 state and logic 1 state thresholds for the signal are not specified on the command card, the default values are used. The default values can be set using the command `usim_opt` `vl` = *value* `vh` = *value.*

Arguments

`title`	The title of the current timing analysis.
`node`	Specifies the name of the node on which the hold timing check is performed. Wildcards are supported in the node name (see Chapter 3, "Simulation Options").
`min_low_time`	The minimum value of the pulse width in logic 0 state.
`max_low_time`	The maximum value of the pulse width in logic 0 state.
`min_high_time`	The minimum value of the pulse width in logic 1 state.
`max_high_time`	The maximum value of the pulse width in logic 1 state.
`vl=`*`logic_0_threshold`*	The threshold of the logic 0 state for the signal. If the signal has value less than `vl`, it is considered to be logic 0.
`vh=`*`logic_1_threshold`*	The threshold of the logic 1 state for the signal. If the signal has value greater than `vh`, it is considered to be logic 1.

`depth=`*`value`*	Specifies the depth in the circuit hierarchy that a wildcard name applies. If it is set as one, only the nodes at the current level is applied (default value is infinity).
`subckt=`*`name`*	Specifies the subcircuit to which this statement applies. By default, it applies to the top level. If the statement is already in a subcircuit definition, this parameter is ignored. Setting this parameter is equivalent to defining the statement within a subcircuit declaration.
`start=time1`	Timing analysis start time. The Virtuoso UltraSim simulator does not perform a timing analysis for simulation `time` < `time1`.
`stop=time2`	Timing analysis end time. The simulator does not perform a timing analysis for simulation `time` > `time2`.

Example

Spectre Syntax:

```
usim_ta ta_all pulsew example node=n1 tmin_low=4n tmax_low=6n tmin_high=5n
tmax_high=8n
```

SPICE Syntax:

```
.usim_ta ta_all pulsew example node=n1 tmin_low=4n tmax_low=6n tmin_high=5n
tmax_high=8n
```

tells the Virtuoso UltraSim simulator to report pulse width errors if node `n1` stays in the logic 0 state for less than 4 ns or longer than 6 ns, or if it stays in the logic 1 state for less than 5 ns or more than 8 ns.

Setup Check

Spectre Syntax

```
usim_ta title setup node=node1 edge=rise|fall|both ref_node=node2
     ref_edge=rise|fall|both setup_time=time [window=window_size]
     [vl=logic_0_threshold] [vh=logic_1_threshold] [vrl=logic_0_threshold]
     [vrh=logic_1_threshold] [depth=value] [subckt=name] [start=time1]
     [stop=time2]
```

SPICE Syntax

```
.usim_ta title setup node=node1 edge=rise|fall|both ref_node=node2
     ref_edge=rise|fall|both setup_time=time [window=window_size]
     [vl=logic_0_threshold] [vh=logic_1_threshold] [vrl=logic_0_threshold]
     [vrh=logic_1_threshold] [depth=value] [subckt=name] [start=time1]
     [stop=time2]
```

Description

This command is used to report setup timing errors on the specified node(s) with respect to a reference node. A setup timing error has occurred if a permissible signal transition occurs between the times `t_ref-setup_time` and `t_ref+window_size`, where `t_ref` is the time when a permissible reference transition occurs.

A permissible transition has occurred if the waveform crosses the corresponding logic threshold. For example, when a waveform crosses the logic 1 threshold while rising, it has a RISE transition. If the logic 0 state and logic 1 state thresholds for the signal, reference, or both, are not specified on the command card, then the values must be set with the command `usim_opt` vl = *value* vh = *value.*

Arguments

`title`	The title of the current timing analysis.
`node`	Specifies the name of the node on which the setup timing check is performed. Wildcards are supported in the node name (see Chapter 3, “Simulation Options”).
`edge=(rise|fall|both)`	The permissible transition type for the signal node. `rise`, a low-to-high transition is the permissible transition. `fall`, a high-to-low transition is the permissible transition. `both`, a low-to-high or a high-to-low transition is a permissible transition.
`ref_node`	The name of the reference node. Only a single node is allowed.

`ref_edge=(rise|fall|both)`	The permissible transition for the reference node. `rise`, a low-to-high transition is the permissible transition. `fall`, a high-to-low transition is the permissible transition. `both`, a low-to-high or a high-to-low transition is a permissible transition.
`setup_time`	The setup time. It can be positive or negative. If negative, the `window` parameter must be specified.
`window=`*`window_size`*	The time window after the reference transition. This parameter must be specified when the setup time is negative.
`vl=`*`logic_0_threshold`*	The threshold of logic 0 state for a signal. If the signal has a value less than `vl`, it is considered to be logic 0.
`vh=`*`logic_1_threshold`*	The threshold of logic 1state for a signal. If the signal has a value greater than `vh`, it is considered to be logic 1.
`vrl=`*`logic_0_threshold`*	The threshold of logic 0 state for a reference. If the reference has a value less than `vrl`, it is considered to be logic 0.
`vrh=`*`logic_1_threshold`*	The threshold of logic 1 state for a reference. If the reference has a value greater than `vrh`, it is considered to be logic 1.
`depth=`*`value`*	Specifies the depth in the circuit hierarchy that a wildcard name applies. If it is set as one, only the nodes at the current level is applied (default value is infinity).
`subckt=`*`name`*	Specifies the subcircuit that this statement applies in. By default, it applies in the top level. If the statement is already in a subcircuit definition, this parameter is ignored. Setting this parameter is equivalent to defining the statement within a subcircuit declaration.
`start=time1`	Timing analysis start time. The Virtuoso UltraSim simulator does not perform a timing analysis for simulation `time` < `time1`.

`stop=time2`	Timing analysis end time. The simulator does not perform a timing analysis for simulation `time` > `time2`.

Examples

In the following Spectre example

```
usim_opt vl=0.3 vh=0.7
```

```
usim_ta ta_all setup example1 node=n1 edge=rise ref_node=clk ref_edge=rise
setup_time=2n
```

tells the Virtuoso UltraSim simulator to report setup timing errors if the `rise` transitions on node `n1` occur within the 2 ns before a `rise` transition on the node `clk`. Since the low and high thresholds are not specified in the command, the values in `usim_opt` are used in the analysis.

In the following SPICE example

```
.usim_ta ta_all setup example2 node=n2 edge=both ref_node=clk ref_edge=fall
setup_time=-1ns window=3ns
```

tells the simulator to report setup timing errors if the `rise` or the `fall` transitions on node `n2` occur within the time interval of 1 ns after the `fall` transition of the node `clk`, and 3 ns after the `clk` `fall` transition.

Timing Edge Check

Spectre Syntax

```
usim_ta title edge node=node1 edge=rise|fall|both ref_node=node2
     ref_edge=rise|fall|both td_min=min_time td_max=max_time
     <vl=logic_0_threshold> <vh=logic_1_threshold> <vrl=logic_0_threshold>
     <vrh=logic_1_threshold> <trigger=trigger_type> <depth=value> <subckt=name>
     <start=time1> <stop=time2>
```

SPICE Syntax

```
.usim_ta title edge node=node1 edge=rise|fall|both ref_node=node2
     ref_edge=rise|fall|both td_min=min_time td_max=max_time
     <vl=logic_0_threshold> <vh=logic_1_threshold> <vrl=logic_0_threshold>
     <vrh=logic_1_threshold> <trigger=trigger_type> <depth=value> <subckt=name>
     <start=time1> <stop=time2>
```

Description

This command is used to report timing edge errors on the specified node(s) with respect to a reference node. A timing edge error occurs when the permissible signal transition time falls outside the range `t_ref+min_time` and `t_ref+max_time`, where `t_ref` is the time that the permissible reference transition occurs.

A permissible transition occurs when the waveform crosses the corresponding logic threshold. For example, when a waveform crosses the logic 1 threshold while rising, it has a RISE transition. If the logic 0 state and logic 1 state thresholds for the signal, reference, or both, are not specified on the command card, the default values are used. The default values can be set using the command `usim_opt` `vl` = *value* `vh` = *value.* The `trigger` option allows you to decide whether a permissible signal transition, a permissible reference transition, or both trigger the timing edge check.

Arguments

`title`	The title of the current timing analysis.
`node`	Specifies the name of the node on which the timing edge check is performed. Wildcards are supported in the node name (see Chapter 3, "Simulation Options").
`edge=(rise|fall|both)`	The permissible transition type for the signal nodes. `rise`, a low-to-high transition is the permissible transition. `fall`, a high-to-low transition is the permissible transition. `both`, a low-to-high or a high-to-low transition is a permissible transition.
`ref_node`	The name of the reference node. Only a single node is allowed.

`ref_edge=(rise|fall|both)`	The permissible transition type for the reference nodes. `rise`, a low-to-high transition is the permissible transition. `fall`, a high-to-low transition is the permissible transition. `both`, a low-to-high or a high-to-low transition is a permissible transition.
`min_time`	The minimum value of the delay between the permissible transitions of the signal and the reference.
`max_time`	The maximum value of the delay between the permissible transitions of the signal and the reference.
`vl=`*`logic_0_threshold`*	The threshold of logic 0 state for a signal. If the signal has a value less than `vl`, it is considered to be logic 0.
`vh=`*`logic_1_threshold`*	The threshold of logic 1 state for a signal. If the signal has a value greater than `vh`, it is considered to be logic 1.
`vrl=`*`logic_0_threshold`*	The threshold of logic 0 state for a reference. If the reference has a value less than `vrl`, it is considered to be logic 0.
`vrh=`*`logic_1_threshold`*	The threshold of logic 1 state for a reference. If the reference has a value greater than `vrh`, it is considered to be logic 1.
`trigger=(ref|sig|both)`	The trigger to start a timing edge check. `ref`, a permissible transition at a reference triggers the check (this is the default value) `sig`, a permissible transition at a signal triggers the check. `both`, a permissible transition if a reference or a signal triggers the check.
`depth=`*`value`*	Specifies the depth in the circuit hierarchy that a wildcard name applies. If it is set as one, only the nodes at the current level is applied (default value is infinity).

`subckt=`*`name`*	Specifies the subcircuit to which this statement applies. By default, it applies to the top level. If the statement is already in a subcircuit definition, this parameter is ignored. Setting this parameter is equivalent to defining the statement within a subcircuit declaration.
`start=time1`	Timing analysis start time. The Virtuoso UltraSim simulator does not perform a timing analysis for simulation `time` < `time1`.
`stop=time2`	Timing analysis end time. The simulator does not perform a timing analysis for simulation `time` > `time2`.

Examples

In the following Spectre example

```
usim_ta ta_all edge example1 node=n1 edge=rise ref_node=clk ref_edge=rise
td_min=2n td_max=5n
```

tells the Virtuoso UltraSim simulator to report timing edge errors if the delay between the `rise` transitions at node `n1` and reference `clk` is less than 2 ns, or longer than 5 ns. Since the default value of `trigger` is `ref`, only a `rise` transition of the reference can trigger a timing edge check.

In the following SPICE example

```
.usim_ta ta_all edge example2 node=n2 edge=rise ref_node=clk ref_edge=rise
td_min=2n td_max=5n trigger=sig
```

tells the simulator to report timing edge errors if the delay is outside the range of 2 ns and 5 ns. In this case, only a `rise` transition of the signal at `n2` can trigger a timing edge check.

Bisection Timing Optimization

Description

When analyzing circuit timing violations and optimizing timing margins, multiple simulations and iterative analysis of the results is required. Virtuoso UltraSim simulator bisection timing optimization combines multiple simulations into a single characterization, reducing characterization time and simplifying the process. Typical applications include cell characterization timing measurements, and setup and hold timing optimization.

Bisection methodology, using a binary search strategy, seeks the optimal value of a specified input parameter associated with the goal value of an output variable. During a bisection search, the Virtuoso UltraSim simulator performs the following steps:

1. Transient simulation with the specified parameter set at the lower and upper limits, respectively.

   The measurement results for the lower and upper limits need to meet the pre-determined goal with one limit, and fail with the other (otherwise the simulator ends the simulation and prints a message).

2. Simulation occurs at the mid-point of the search range and the resulting measurement is compared with the goal value.

   The search range can be split into halves by choosing a new search range. The measurement results determine whether the first or second half is used in the search range.

3. Multiple simulations occur until one of the following conditions are met: The relative tolerance for the input and output variables is satisfied or the maximum number of iterations is reached.

The bisection timing optimization feature only applies to vsource, and the sweeping parameter must be included in the expression of delay for either the pulse function or time-value pairs in the pwl function. If the Virtuoso UltraSim simulator is unable to find a qualified vsource, the bisection feature is turned off.

To use the simulator for bisection timing optimization, more accurate settings than the default simulator settings may be required. Cadence recommends first evaluating which Virtuoso UltraSim `sim_mode` and `speed` fulfils the specific accuracy requirements of your design before using bisection timing optimization (see Chapter 3, “Simulation Options” for more information about `sim_mode` and `speed`).

The Virtuoso UltraSim simulator reports the search process and optimized parameters in the `.optlog` file, and also generates waveforms and the measurement result (`.mt0`) for the final simulation.

Arguments

The following statements can be used for model optimization (.model), parameter optimization (.param), measurement (.measure), and transient analysis (.tran).

.model

```
.model optmodelname opt method=bisection <relin=value> <relout=value>
     <itropt=value>
```

This statement defines the optimization method (bisection) and the criteria used to determine the maximum number of iterations, relative tolerances, and to stop an iteration.

`optmodelname`	Name of the optimization model
`method`	Defines the optimization method **Note:** Only bisection is supported.
`relin`	Specifies the relative tolerance of the input parameter (default value is 0.001)
`relout`	Specifies the relative tolerance of the output variable (default value is 0.001)
`itropt`	Specifies the maximum number of iterations (default value is 20 iterations)

.param

```
.param paramname=optxxxx(<initial>, <lower>, <upper>)
```

This statement defines the optimization parameter (input variable) and its initial value, and the lower and upper limit values. The bisection method allows only one parameter and ignores the initial value of the parameter.

`paramname`	Name of the optimization parameter

`initial`	Specifies the initial value of the parameter **Note:** The initial value is not used for bisection.
`lower`	Specifies the lower limit of the parameter
`upper`	Specifies the upper limit of the parameter

.measure

```
.measure tran meastitle <measfuncs> goal=goalvalue
```

This statement defines the measurement of the output variable and its goal value, which is used by the Virtuoso UltraSim simulator to evaluate the validity of a parameter value (that is, determines whether or not the parameter value is accepted in the analysis).

`meastitle`	Name of the statement
`measfuncs`	Specifies the measurement functions supported in a base-level `.measure` statement
`goal`	Specifies the desired value of the measurement

.tran

```
.tran <transtep> <tranendtime> sweep optimize=optparfun results=meastitle
     model=optmodename fastsweep=on|off
```

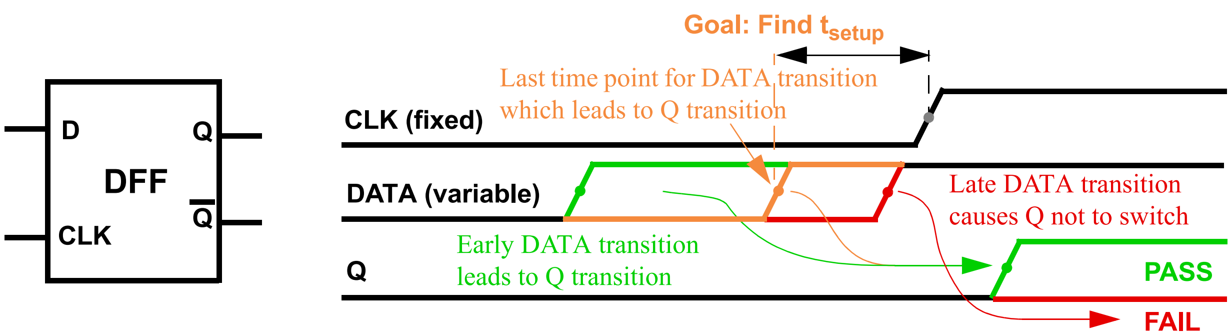
This statement defines the bisection and optimization methods.

`optparfun`	Name of the parameter function given in the `.param` statement.
`meastitle`	Name of the `.measure` statement.
`optmodelname`	Name of the optimization model given in the `.model` statement.
`fastsweep`	**off** – netlist file is parsed and simulation database is rebuilt for each bisection iteration (default). **on** – netlist file parsing and building of simulation database is skipped for all bisection iterations after the first iteration (simulation database from first iteration is reused). This feature provides a performance advantage, but is limited to bisection applications with no change in the topology and initial circuit conditions between iterations.

Example

The following example illustrates how to measure the setup time of a delay-type flip flop (D-FF).

Figure 8-3 D-FF Setup Time Optimization



The D-FF has two input signals (`DATA` and `CLK`) and two output signals (`Q` and `Q_`). The assumption is that both input signals transition (0->1) at `Td` and `Tclk`, respectively. It is expected that the data will remain stable during setup time, until `CLK` switches.

The transition needs to satisfy the following condition,

```
Tclk > Td + setup_time
```

In this case, a transition (0->1) at the output of the D-FF `Q` occurs. Otherwise, no transition is found by the simulator and output `Q` remains at 0. The transition at the output can be detected by measuring the `max` value at `Q`. If the measurement result is 1, there is a transition; if 0, no transition occurs.

The following is a sample top-level netlist file containing bisection timing optimization settings.

```
**** Search setup time for D-FF by bisection method ****
.param vdd=2.5
...
// PWL stimulus for CLK & data
// td=delay characterizes the setup time and is to be adjusted by bisection
Vclk CLK 0 pwl(0n 0 1n 0 1.5n vdd 3n vdd 3.5n 0 10n 0 10.5n vdd)
Vdata data 0 pwl(0n 0 5n 0 5.5n vdd td=delay)
// instance of D Flip-Flop
x1 data CLK Q_ Q DFF_B
// set delay to be the input variable, and its searching range
.param delay=opt1(0n, 0n, 6n)
// set optimization method to be bisection
.model optmod opt method=bisection
```

```
// set measurement to find the transition of output and its goal value
.measure tran vout max v(Q) goal='0.9*vdd'
// set bisection transient simulation
.tran 0.1n 20n sweep optimize=opt1 results=vout model=optmod
// measure setup time
.measure tran setup_time trig v(data) value='0.5*vdd' rise=1 td=5n
+ targ v(clk) value='0.5*vdd' rise=1 td=5n
.end
```

The following sample output file shows the simulation results. The optimized value of delay is 5.083 ns. With this delay, the voltage at the `Q` output of the DFF is 2.504 v.

```
iter          lower         upper          current        result
1             0             6e-09          0              2.50407
2             0             6e-09          6e-09          0.013865
3             0             6e-09          3e-09          2.50407
4             3e-09         6e-09          4.5e-09        2.50398
5             4.5e-09       6e-09          5.25e-09       0.0138437
6             4.5e-09       5.25e-09       4.875e-09      2.5046
7             4.875e-09     5.25e-09       5.0625e-09     2.5043
8             5.0625e-09    5.25e-09       5.15625e-09    0.0138437
9             5.0625e-09    5.15625e-09    5.10938e-09    0.0138437
10            5.0625e-09    5.10938e-09    5.08594e-09    0.0263151
11            5.0625e-09    5.08594e-09    5.07422e-09    2.50424
12            5.07422e-09   5.08594e-09    5.08008e-09    2.50422
13            5.08008e-09   5.08594e-09    5.08301e-09    2.50396
Optimization  Method        bisection
Optimization  Parameter     delay
Optimized Value             5.08301e-09
vout                        2.50396
Goal                        2.25
```

Static Checks

The Virtuoso UltraSim simulator provides static checks which can be used to analyze circuit topology, parameter values, simulation information, and device and element characteristics.

Note: Static checks can be performed without a transient analysis by removing the transient analysis statement from the netlist file.

- Netlist File Parameter Check checks whether the element size and simulation temperature are in the reasonable range or not.
- Print Parameters in Subcircuit prints the parameters located in a subcircuit.
- Resistor and Capacitor Statistical Checks determines whether resistor or capacitor values are within a specified range.
- Substrate Forward-Bias Check checks whether a MOSFET substrate becomes forward-biased.
- Static MOS Voltage Check monitors whether MOSFET bias voltage exceeds specified bounds or conditions.
- Static NMOS and PMOS Bulk Forward-Bias Checks determines whether bulk to drain/source junctions of NMOSFETs or PMOSFETs become forward-biased.
- Detect Conducting NMOSFETs and PMOSFETs checks if the NMOSFET minimum gate voltage is greater than the minimum specified for drain/source voltages or the PMOSFET maximum gate voltage is lower than the maximum specified for drain/source voltages.
- Static Maximum Leakage Path Check detects DC leakage paths between all voltage sources.
- Static High Impedance Check detects high impedance nodes without running DC or transient simulations.
- Static ERC Check detects electrical design rule violations without running DC or transient simulations.
- Static DC Path Check detects a DC path between voltage sources without running DC or transient simulation.
- info Analysis gives access to input/output values and operating-point parameters.
- Partition and Node Connectivity Analysis used for debugging (for example, checking the size of partitions and node connectivity).
- Warning Message Limit Categories customizes how warning messages are handled by the Virtuoso UltraSim simulator.

Netlist File Parameter Check

Spectre Syntax

```
usim_report chk_param <ermaxcap=v> <ermaxres=v> <ermaxmosw=v> <ermaxmosl=v>
     <ermaxmosas=v> <ermaxmosps=v> <ermaxmostox=v> <ermaxdiodew=v>
     <ermaxdiodel=v> <ermaxdiodea=v> <ermaxtemp=v> <erminfactor=v> <ermincap=v>
     <erminres=v> <erminmosw=v> <erminmosl=v> <ermaxmosad=v> <ermaxmospd=v>
     <erminmostox=v> <ermindiodew=v> <ermindiodel=v> <ermindiodea=v>
     <ermintemp=v> <model=m> <wamaxcap=v> <wamaxres=v> <wamaxmosw=v> <wamaxmosl=v>
     <wamaxmosas=v> <wamaxmosps=v> <wamaxmostox=v> <wamaxdiodew=v>
     <wamaxdiodel=v> <wamaxdiodea=v> <wamaxtemp=v> <waminfactor=v> <wamincap=v>
     <waminres=v> <waminmosw=v> <waminmosl=v> <wamaxmosad=v> <wamaxmospd=v>
     <waminmostox=v> <wamindiodew=v> <wamindiodel=v> <amindiodea=v> <wamintemp=v>
```

SPICE Syntax

```
.usim_report chk_param <ermaxcap=v> <ermaxres=v> <ermaxmosw=v> <ermaxmosl=v>
     <ermaxmosas=v> <ermaxmosps=v> <ermaxmostox=v> <ermaxdiodew=v>
     <ermaxdiodel=v> <ermaxdiodea=v> <ermaxtemp=v> <erminfactor=v> <ermincap=v>
     <erminres=v> <erminmosw=v> <erminmosl=v> <ermaxmosad=v> <ermaxmospd=v>
     <erminmostox=v> <ermindiodew=v> <ermindiodel=v> <ermindiodea=v>
     <ermintemp=v> <model=m> <wamaxcap=v> <wamaxres=v> <wamaxmosw=v> <wamaxmosl=v>
     <wamaxmosas=v> <wamaxmosps=v> <wamaxmostox=v> <wamaxdiodew=v>
     <wamaxdiodel=v> <wamaxdiodea=v> <wamaxtemp=v> <waminfactor=v> <wamincap=v>
     <waminres=v> <waminmosw=v> <waminmosl=v> <wamaxmosad=v> <wamaxmospd=v>
     <waminmostox=v> <wamindiodew=v> <wamindiodel=v> <amindiodea=v> <wamintemp=v>
```

Description

The `chk_param` command checks whether or not the element sizes and simulation temperatures are within a reasonable range. This command is executed after the netlist file is parsed, and the Virtuoso UltraSim simulator generates a report file with a `.rpt_chkpar` suffix.

When the checked data exceeds specified or default soft upper/lower limits, warning messages are issued. If the data abnormality exceeds specified or default absolute limits, error messages are generated and the simulation stops. Multiple `chk_param` command lines are supported by the Virtuoso UltraSim simulator.

Note: All errors are collected and printed by `chk_param`, and then the simulation is stopped (that is, `chk_param` is always performed on all instance parameters). If optional arguments are specified, the related parameters are checked using the specified value and the remaining parameters are checked using the default values.

Arguments

`ermaxcap=v`	The Virtuoso UltraSim simulator issues an error message and stops the simulation if any capacitance exceeds the specified upper bound value (default value is 1.0e-3 F)
`wamaxcap=v`	The simulator issues a warning message and continues the simulation if any capacitance exceeds the specified upper bound value (default value is 1.0e-8 F)
`ermincap=v`	The simulator issues an error message and stops the simulation if any capacitance is less than the specified lower bound value (default value is -1.0e-15 F)
`wamincap=v`	The simulator issues a warning message and continues the simulation if any capacitance is less than the specified lower bound value (default value is 0) **Note:** The value needs to meet the following criteria: `ermincap` <= `wamincap` <= `wamaxcap` <= `ermaxcap` (otherwise a warning message is issued and the default value is used instead).
`ermaxres=v`	The simulator issues an error message and stops the simulation if any resistance exceeds the specified upper bound value (default value is 1.0e+15 ohms)
`wamaxres=v`	The simulator issues a warning message and continues the simulation if any resistance exceeds the specified upper bound value (default value is 1.0e+12 ohms)
`erminres=v`	The simulator issues an error message and stops the simulation if any resistance is less than the specified lower bound value (default value is 0)

`waminres=v`	The simulator issues a warning message and continues the simulation if any resistance is less than the specified lower bound value (default value is 0) **Note:** The value needs to meet the following criteria: `erminres` <= `waminres` <= `wamaxres` <= `ermaxres` (otherwise a warning message is issued and the default value is used instead).
`ermaxmosw=v`	The simulator issues an error message and stops the simulation if any MOSFET channel width exceeds the specified upper bound value (default value is 1.0e-2 m)
`wamaxmosw=v`	The simulator issues a warning message and continues the simulation if any MOSFET channel width exceeds the specified upper bound value (default value is 1.0e-3 m)
`erminmosw=v`	The simulator issues an error message and stops the simulation if any MOSFET channel width is less than the specified lower bound value (default value is 1.0e-8 m)
`waminmosw=v`	The simulator issues a warning message and continues the simulation if any MOSFET channel width is less than the specified lower bound value (default value is 1.0e-7 m) **Note:** The value needs to meet the following criteria: `erminmosw` <= `waminmosw` <= `wamaxmosw` <= `ermaxmosw` (otherwise a warning message is issued and the default value is used instead).
`ermaxmosl=v`	The simulator issues an error message and stops the simulation if any MOSFET channel length exceeds the specified upper bound value (default value is 1.0e-2 m)
`wamaxmosl=v`	The simulator issues a warning message and continues the simulation if any MOSFET channel length exceeds the specified upper bound value (default value is 1.0e-3 m)

`erminmosl=v`	The simulator issues an error message and stops the simulation if any MOSFET channel length is less than the specified lower bound value (default value is 1.0e-8 m)
`waminmosl=v`	The simulator issues a warning message and continues the simulation if any MOSFET channel length is less than the specified lower bound value (default value is 1.0e-7 m) **Note:** The value needs to meet the following criteria: `erminmosl` <= `waminmosl` <= `wamaxmosl` <= `ermaxmosl` (otherwise a warning message is issued and the default value is used instead).
`ermaxmosad=v`	The simulator issues an error message and stops the simulation if any MOSFET drain diffusion area exceeds the specified upper bound value (default value is 1.0e-4 m^2)
`wamaxmosad=v`	The simulator issues a warning message and continues the simulation if any MOSFET drain diffusion area exceeds the specified upper bound value (default value is 1.0e-6 m^2) **Note:** The value needs to meet the following criteria: `wamaxmosad` <= `ermaxmosad` (otherwise a warning message is issued and the default value is used instead).
`ermaxmosas=v`	The simulator issues an error message and stops the simulation if any MOSFET source diffusion area exceeds the specified upper bound value (default value is 1.0e-4 m^2)
`wamaxmosas=v`	The simulator issues a warning and continues the simulation if any MOSFET source diffusion area exceeds the specified upper bound value (default value is 1.0e-6 m^2) **Note:** The value needs to meet the following criteria: `wamaxmosas` <= `ermaxmosas` (otherwise a warning message is issued and the default value is used instead).

`ermaxmospd=v`	The simulator issues an error message and stops the simulation if any MOSFET perimeter of the drain junction exceeds the specified upper bound value (default value is 1.0e-2 m^2)
`wamaxmospd=v`	The simulator issues a warning message and continues the simulation if any MOSFET perimeter of the drain junction exceeds the specified upper bound value (default value is 1.0e-3 m^2) **Note:** The value needs to meet the following criteria: `wamaxmospd` <= `ermaxmospd` (otherwise a warning message is issued and the default value is used instead).
`ermaxmosps=v`	The simulator issues an error message and stops the simulation if any MOSFET perimeter of the source junction exceeds the specified upper bound value (default value is 1.0e-2 m^2)
`wamaxmosps=v`	The simulator issues a warning message and continues the simulation if any MOSFET perimeter of the source junction exceeds the specified upper bound value (default value is 1.0e-3 m^2) **Note:** The value needs to meet the following criteria: `wamaxmosps` <= `ermaxmosps` (otherwise a warning message is issued and the default value is used instead).
`ermaxmostox=v`	The simulator issues an error message and stops the simulation if any MOSFET gate oxide thickness exceeds the specified upper bound value (default value is 5.0e-8 m)
`wamaxmostox=v`	The simulator issues a warning message and continues the simulation if any MOSFET gate oxide thickness exceeds the specified upper bound value (default value is 3.0e-8 m)
`erminmostox=v`	The simulator issues an error message and stops the simulation if any MOSFET gate oxide thickness is less than the specified lower bound value (default value is 5.0e-10 m)

`waminmostox=v`	The simulator issues a warning message and continues the simulation if any MOSFET gate oxide thickness is less than the specified lower bound value (default value is 5.0e-9 m) **Note:** The value needs to meet the following criteria: `erminmostox` <= `waminmostox` <= `wamaxmostox` <= `ermaxmostox` (otherwise a warning message is issued and the default value is used instead).
`ermaxdiodew=v`	The simulator issues an error message and stops the simulation if any diode width exceeds the specified upper bound value (default value is 1.0e-2 m)
`wamaxdiodew=v`	The simulator issues a warning message and continues the simulation if any diode width exceeds the specified upper bound value (default value is 1.0e-3 m)
`ermindiodew=v`	The simulator issues an error message and stops the simulation if any diode width is less than the specified lower bound value (default value is 1.0e-8 m)
`wamindiodew=v`	The simulator issues a warning message and continues the simulation if any diode width is less than the specified lower bound value (default value is 1.0e-7 m) **Note:** The value needs to meet the following criteria: `ermindiodew` <= `wamindiodew` <= `wamaxdiodew` <= `ermaxdiodew` (otherwise a warning message is issued and the default value is used instead).
`ermaxdiodel=v`	The simulator issues an error message and stops the simulation if any diode length exceeds the specified upper bound value (default value is 1.0e-2 m)
`wamaxdiodel=v`	The simulator issues a warning message and continues the simulation if any diode length exceeds the specified upper bound value (default value is 1.0e-3 m)
`ermindiodel=v`	The simulator issues an error message and stops the simulation if any diode length is less than the specified lower bound value (default value is 1.0e-8 m)

`wamindiodel=v`	The simulator issues a warning message and continues the simulation if any diode length is less than the specified lower bound value (default value is 1.0e-7 m) **Note:** The value needs to meet the following criteria: `ermindiodel` <= `wamindiodel` <= `wamaxdiodel` <= `ermaxdiodel` (otherwise a warning message is issued and the default value is used instead).
`ermaxdiodea=v`	The simulator issues an error message and stops the simulation if any diode area exceeds the specified upper bound value (default value is 1.0e-4 m^2)
`wamaxdiodea=v`	The simulator issues a warning message and continues the simulation if any diode area exceeds the specified upper bound value (default value is 1.0e-6 m^2)
`ermindiodea=v`	The simulator issues an error message and stops the simulation if any diode area is less than the specified lower bound value (default value is 1.0e-16 m^2)
`wamindiodea=v`	The simulator issues a warning message and continues the simulation if any diode area is less than the specified lower bound value (default value is 1.0e-14 m^2) **Note:** The value needs to meet the following criteria: `ermindiodea` <= `wamindiodea` <= `wamaxdiodea` <= `ermaxdiodea` (otherwise a warning message is issued and the default value is used instead).
`ermaxtemp=v`	The simulator issues an error message and stops the simulation if the circuit temperature, in degrees Celsius, exceeds the specified upper bound value (default value is 500)
`wamaxtemp=v`	The simulator issues a warning message and continues the simulation if the circuit temperature, in degrees Celsius, exceeds the specified upper bound value (default value is 150)
`ermintemp=v`	The simulator issues an error message and stops the simulation if the circuit temperature, in degrees Celsius, is less than the specified lower bound value (default value is -200)

`wamintemp=v`	The simulator issues a warning message and continues the simulation if the circuit temperature, in degrees Celsius, is less than the specified lower bound value (default value is -100) **Note:** The value needs to meet the following criteria: `ermintemp` <= `wamintemp` <= `wamaxtemp` <= `ermaxtemp` (otherwise a warning message is issued and the default value is used instead).
`ermaxfactor=v`	The simulator issues an error message and stops the simulation if any instance multiplier is larger than the specified upper bound value.
`wamaxfactor=v`	The simulator issues a warning message and continues the simulation if any instance multiplier factor is larger than the specified upper bound value.
`erminfactor=v`	The simulator issues an error message and stops the simulation if any instance multiplier is less than the specified lower bound value.
`waminfactor=v`	The simulator issues a warning message and continues the simulation if any instance multiplier factor is less than the specified lower bound value. **Note:** The value needs to meet the following criteria: `erminfactor` <= `waminfactor <= wamaxfactor <= ermaxfactor` (a warning message is issued if the criteria is not met).
`model=m`	Defines the model card name; if specified, all instances of the specified model name have their specific values checked (optional). **Note:** If the `model` argument is not specified, the check is applied to all instances.

Example 1

In the following Spectre example

```
usim_report chk_param
```

The Virtuoso UltraSim simulator checks all instance parameters in the netlist file to make sure they are within the default value limits. If the values exceed the limits, the simulator issues warning or error messages.

Example 2

In the following SPICE example

```
.usim_report chk_param ermaxmosl=2u wamaxmosl=1u erminmosl=0.09u waminmosl=0.1u
```

The simulator checks all instance parameters in the netlist file to make sure they are within the default value limits (uses specified values to check the channel length of all MOSFET models). For example, if the channel length of any MOSFET model is greater than 2 um or less than 0.09 um, the simulator stops the simulation and issues an error message (if the channel length is greater than 1 um or less than 0.1 um, the simulation continues and a warning message is issued).

Example 3

In the following Spectre example

```
usim_report chk_param ermaxmosl=2u wamaxmosl=1u erminmosl=0.09u waminmosl=0.1u
model=hvmos
```

The simulator checks all instance parameters in the netlist file to make sure they are within the default value limits (uses specified values to check the channel length of all HVMOS MOSFET instances).

The following is an example of a `xxxx.rpt_chkpar` report file.

```
****** Parameters Check Errors ******
Total of 2 error(s) reported.

Model       Subckt    Parameter     Limits          Instance
resistor    ---       r = -0.12     ( < 0.0 )       r23
mos1        por       l = 1.0e-9    ( < 1.0e-8 )    xtop.xpor.m100

****** Parameters Check Warnings*******
Total of 2 warning(s) reported.

Model       Subckt    Parameter     Limits          Instance
diode1      bg        w = 0.002     ( < 0.003 )     x0.x1.x2.x3.d4
capacitor   ---       c = 1.0e-7    ( > 1.0e-8 )    c1

****** End of Parameter Check. ******
```

Print Parameters in Subcircuit

Spectre Syntax

```
usim_report param param_name [depth=..]
```

SPICE Syntax

```
.usim_report param param_name [depth=..]
```

Description

This option enables you to print subcircuit parameters into a `netlist.para_rpt` file. The option also supports wildcards and allows use of the `depth` argument to limit the levels of hierarchy (default for `depth` is infinity). Matching is case insensitive.

For more information about wildcards, see "Wildcard Rules" on page 55.

Examples

In the following Spectre example

```
usim_report param *
```

tells the simulator to print out all of the parameters from the entire design hierarchy.

In the following SPICE example

```
.usim_report param * depth=1
```

tells the simulator to print out all of the top level parameters.

In the following Spectre example

```
usim_report param x1.a
```

tells the simulator to look for a parameter named `a` in instance `x1`. If no match is found, the simulator issues a warning message.

In the next example

```
usim_report param x1.*
```

tells the simulator to print out all of the parameters in instance `x1` and all the instances below `x1`.

In the next example

```
usim_report param x1*.* depth=2
```

tells the simulator to print out all of the parameters in all instances that are two hierarchies lower and have instance names that start with `x1`.

Examples of instance names starting with `x1` include `x1.aa`, `x1a.b`, `x1.x2.bb`, and `x1b.x3.bb` (`aa`, `b`, and `bb` are the parameter names).

Resistor and Capacitor Statistical Checks

Resistor Statistical Check

Spectre Syntax

```
usim_report resistor type=warning rmax=value
usim_report resistor type=distr rmin=value rmax=value
usim_report resistor type=print rmin=value rmax=value nlimit=num sort=[dec|inc]
```

SPICE Syntax

```
.usim_report resistor type=warning rmax=value
.usim_report resistor type=distr rmin=value rmax=value
.usim_report resistor type=print rmin=value rmax=value nlimit=num sort=[dec|inc]
```

Description

The Virtuoso UltraSim simulator resistor statistical check determines if resistor values are within a reasonable user-defined range.

- **type=warning** prints a warning about small resistors and reports the number of resistors with values below `rmax`.
- **type=distr** prints resistor statistics into a `*xxxx*.chk_resistor` log file for resistors with values between `rmin` and `rmax`.
- **type=print** prints resistors with values between `rmin` and `rmax` into a `*xxxx*.chk_resistor` log file.

Arguments

`type=warning|distr|print`	Type of resistor statistic.
`rmax=value`	Specifies upper bound of resistor value to be reported (default value is 0.1 ohms).
`rmin=value`	Specifies lower bound of resistor value to be reported (default value is 0).

`nlimit=num`	Limits number of resistors printed in report (integer; default value is 10 resistors).
`sort=dec|inc`	Specifies sorting order printed resistors. If set to `inc`, resistors are sorted in increasing order of their values (default). If set to `dec`, resistors are sorted in decreasing order of their values.

Example 1

In the following Spectre example

```
usim_report resistor type=warning rmax=0.001
```

The Virtuoso UltraSim simulator issues a warning about small resistors if the circuit contains resistors with values less than 0.001 ohms, and reports the number of resistors in a log file.

Example 2

In the following SPICE example

```
.usim_report resistor type=distr rmin=0 rmax=0.02
```

The simulator generates statistics for resistors with values between 0 and 0.02 ohms in a *`xxxx`*`.chk_resistor` log file.

Example 3

In the following Spectre example

```
usim_report resistor type=print rmin=0 rmax=0.02 nlimit=30 sort=dec
```

The simulator prints the resistors with values between 0 and 0.02 ohms in a *`xxxx`*`.chk_resistor` log file. The resistors are sorted in decreasing order of their value. A total of 30 resistors are printed because `nlimit=30`.

The following is a sample *`xxxx`*`.chk_resistor` log file (includes resistor names and values):

```
.TITLE 'This file is :./test.chk_resistor'
.Usim_report resistor type=distr rmin=0 rmax=0.02
0 - 1m            0
1m - 10m          0
10m - 0.1         3
0.1 - 1           0
.Usim_report resistor type=print rmin=0 rmax=0.02 nlimit=30 sort=dec
```

```
x1.r12          0.001
r05             0.01
r03             0.01
```

Capacitor Statistical Check

Spectre Syntax

```
usim_report capacitor type=warning cmax=value
usim_report capacitor type=distr cmin=value cmax=value
usim_report capacitor type=print cmin=value cmax=value nlimit=num sort=[dec|inc]
```

SPICE Syntax

```
.usim_report capacitor type=warning cmax=value
.usim_report capacitor type=distr cmin=value cmax=value
.usim_report capacitor type=print cmin=value cmax=value nlimit=num sort=[dec|inc]
```

Description

The Virtuoso UltraSim simulator capacitor statistical check determines if capacitor values are within a reasonable user-defined range.

- **type=warning** prints a warning about small capacitors and reports the number of capacitors with values below `cmax`.
- **type=distr** prints capacitor statistics into a `*xxxx*.chk_capacitor` log file for capacitors with values between `cmin` and `cmax`.
- **type=print** prints capacitors with values between `cmin` and `cmax` into a `*xxxx*.chk_capacitor` log file.

Arguments

`type=warning|distr|print`	Type of capacitor statistic.
`cmax=value`	Specifies upper bound of capacitor value to be reported (default value is 1e-16 F).

`cmin=value`	Specifies lower bound of capacitor value to be reported (default value is 0).
`nlimit=num`	Limits number of capacitors printed in report (integer; default value is 10 capacitors).
`sort=dec|inc`	Specifies sorting order for printed capacitors. If set to `inc`, capacitors are sorted in increasing order of their values (default). If set to `dec`, capacitors are sorted in decreasing order of their values.

Example 1

In the following Spectre example

```
usim_report capacitor type=warning cmax=1e-17
```

The Virtuoso UltraSim simulator issues a warning about small capacitors if the circuit contains capacitors with values less than 0.01f F, and reports the number of capacitors in a log file.

Example 2

In the following SPICE example

```
.usim_report capacitor type=distr cmin=0 cmax=1e-17
```

The simulator generates statistics for capacitors with values between 0 and 0.01f F in a *`xxxx`*`.chk_capacitor` log file.

Example 3

In the following Spectre example

```
usim_report capacitor type=print cmin=0 cmax=1e-17 nlimit=30 sort=dec
```

The simulator prints the capacitors with values between 0 and 0.01f F in a *`xxxx`*`.chk_capacitor` log file. The capacitors are sorted in decreasing order of their value. A total of 30 capacitors are printed because `nlimit=30`.

The following is a sample *`xxxx`*`.chk_capacitor` log file (includes capacitor names and values):

```
.TITLE 'This file is :./pump.chk_resistor'
.Usim_report capacitor type=print cmin=0 cmax=20p nlimit=10 sort=dec
x5.c1i1680              1e-11
x5.c1i1679              1e-11
```

```
x5.c1i1678              1e-11
x5.c1i1677              1e-11
x5.c1i1676              1e-11
x5.c1i1675              1e-11
cl                      1e-11
```

Substrate Forward-Bias Check

Spectre Syntax

```
usim_report chk_substrate title mode=[2|1|0] <num=n> <vt=v> <ith=iv> <tth=tv>
     <start=t1> <stop=t2> <model=m>
```

SPICE Syntax

```
.usim_report chk_substrate title mode=[2|1|0] <num=n> <vt=v> <ith=iv> <tth=tv>
     <start=t1> <stop=t2> <model=m>
```

Description

The `chk_substrate` command is used to check if a MOSFET substrate becomes forward-biased. The Virtuoso UltraSim simulator generates a report file with a `.rpt_chksubs` suffix (for example, if the netlist file name is `circuit.sp`, the report is named `circuit.rpt_chksubs`).

Note: This command can only be used to check MOSFET substrates, not other PN junctions.

You can perform substrate forward-bias checking before DC initialization or during the transient simulation. When used before DC initialization, if a PMOS field effect transistor (FET) substrate is connected to a ground or negative voltage source, or a NMOS FET is connected to a positive voltage source, the simulator issues a warning message. The voltage source can be a constant, PWL, pulse, exponential, or sine type. The voltage value at time zero is used by the simulator for substrate forward-bias checking.

When used during transient simulation, if the MOSFET substrate junction is forward-biased more than a threshold voltage (vt) and the junction current is more than a threshold current (ith), a warning message is generated.

Arguments

`title`	Reports titles of warning messages. If the title is not specified, a warning message is issued and the simulator does not perform the check. **Note:** The `title` argument only applies to transient simulations.

`mode=2|1|0`	Specifies checking mode. If the mode is not specified, a warning message is issued and the simulator does not perform the check. ■ **0** – all MOSFET substrate connections are checked before DC initialization. A warning message is printed if a PMOS FET substrate is connected to negative voltage source or a NMOS FET is connected to a positive voltage source. ■ **1** – all MOSFET substrate connections are checked during the transient simulation. ■ **2** – all MOSFET substrate connections are checked before DC initialization and during the transient simulation.
`num`	Specifies maximum number of warning messages issued (default number is 1000 messages).
`vt`	Specifies threshold voltage (default value is 0.5 v). **Note:** The `vt` argument only applies to transient simulations.
`ith`	Specifies threshold current (default value is 0 A). **Note:** The `ith` argument only applies to transient simulations.
`tth`	Duration time (default value is 5 ns).
`start`	Specifies checking start time (default value is 0). **Note:** The `start` argument only applies to transient simulations.
`stop`	Specifies checking stop time (default value is transient stop time). **Note:** The `stop` argument only applies to transient simulations.
`model`	Specifies the MOSFET model names to be checked. When the `model` argument is used, the `vt` and `ith` values apply to MOSFETs for the specified model (all MOSFET instances of remaining models are checked using default values).

Example 1

In the following Spectre example

```
usim_report chk_substrate sub1 mode=0
```

The Virtuoso UltraSim simulator checks the substrates for all the MOSFET models to see if any are forward-biased (checks performed before DC initialization). All warnings issued by the simulator are named `sub1` in the `.rpt_chksubs` file.

Example 2

In the following SPICE example

```
.usim_report chk_substrate sub2 mode=2 vt=0.15
```

The simulator checks the substrates for all the MOSFET models before DC initialization and during the transient simulation. If any MOSFET substrate PN junctions are forward-biased by an amount greater than 0.15 v, `sub2` warnings are issued by the simulator.

Example 3

In the following Spectre example

```
usim_report chk_substrate subscheck3 mode=1 vt=0.6
usim_report chk_substrate subscheck4 mode=1 vt=0.5 model=lvmos
```

The simulator checks the substrates for all the MOSFET models during the transient simulation. If any MOSFET substrate PN junctions are forward-biased by an amount greater than 0.6 v, `subscheck3` warnings are issued by the simulator. If the LVMOS model MOSFETs are forward-biased more than 0.5 v, `subscheck4` warnings are issued.

Here is an example of a `.rpt_chksubs` report file.

```
****** MOS Substrate Forward Biased Before DC ******
Total of 2 Warnings reported.

Model  Subckt vb           Source  Instance
nmos1  or     3.0000e+00   vddh    xtop.xor.m100
pmos2  or     -2.0000e+00  vss     xtop.xor.m101

****** MOS Substrate Forward Biased During Simulation *****
Total of 2 Warnings reported.

Title  Model  Subckt  Time   Vb          Vs          Vd          Instance
sub1   n1     por     3.0e-9 1.8000e+00 1.9387e+00 1.9629e+00 xtop.xpor.m100
sub2   p1     amp     3.0e-6 1.8000e+00 1.9997e+00 1.9507e+00 xtop.xamp.m200
```

Static MOS Voltage Check

Spectre Syntax

```
usim_report chk_mosv title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1
     xinst2 …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1
     skipinst2 …]> <maxmos=n> <vhth=volt> <vlth=volt> <vnth=volt> <vpth=volt>
     <vgdl=volt> <vgdu=volt> <vgsl=volt> <vgsu=volt> <vgbl=volt> <vgbu=volt>
     <vdsl=volt> <vdsu=volt> <vdbl=volt> <vdbu=volt>  <vsbl=volt> <vsbu=volt>
     <cond=expression> <num=n> <rpt_path=0|1> <rpt_node=0|1> <ppos_nneg=0|1>
     <pwl_time=time> <vsrc=[elem1 vmin1 vmax1 elem2 vmin2 vmax2 …]>
     <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]> <xt_vsrc=0|1>
```

SPICE Syntax

```
.usim_report chk_mosv title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1
     xinst2 …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1
     skipinst2 …]> <maxmos=n> <vhth=volt> <vlth=volt> <vnth=volt> <vpth=volt>
     <vgdl=volt> <vgdu=volt> <vgsl=volt> <vgsu=volt> <vgbl=volt> <vgbu=volt>
     <vdsl=volt> <vdsu=volt> <vdbl=volt> <vdbu=volt>  <vsbl=volt> <vsbu=volt>
     <cond=expression> <num=n> <rpt_path=0|1> <rpt_node=0|1> <ppos_nneg=0|1>
     <pwl_time=time> <vsrc=[elem1 vmin1 vmax1 elem2 vmin2 vmax2 …]>
     <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]> <xt_vsrc=0|1>
```

Description

This command is used to check MOSFET bias voltage after the netlist file is parsed, and generates a report if the voltages exceed the specified upper and lower bounds, or meet the specified conditions. The report file format is `xxxx.rpt_chkmosv`.

Arguments

`title`	Title of report.
`model=model_name`	Specifies the model to be checked. The voltage check is applied to transistors with model card name `model_name`.
`subckt`	Specifies the subcircuits to be checked. The voltage check is applied to transistors belonging to subcircuits. Wildcard (`*`) characters can be used.

`inst`	Specifies the instances to be checked. The voltage check is applied to transistors belonging to instances (wildcard characters can be used).
`xsubckt`	Specifies the subcircuits not to be checked (wildcard characters can be used).
`xinst`	Specifies the instances not to be checked (wildcard characters can be used).
`skipsubckt`	Specifies that elements inside designated subcircuits skip voltage propagation (wildcard characters can be used).
`skipinst`	Specifies that elements inside designated instances skip voltage propagation (wildcard characters can be used).
`maxmos=n`	Maximum number of MOSFETs in the voltage propagation path between the voltage source and the MOSFET terminals (default is infinity).
`vhth=volt`	If specified, voltage propagation starts only from constant voltage sources with values greater than or equal to `vhth`.
`vlth=volt`	If specified, voltage propagation starts only from constant voltage sources with values lower than or equal to `vlth`.
`vnth=volt`	NMOSFET threshold voltage. This value is used to calculate the voltage drop across a NMOSFET channel during voltage propagation (default value is 0.5 v).
`vpth=volt`	PMOSFET threshold voltage. This value is used to calculate the voltage drop across a PMOSFET channel during voltage propagation (default value is -0.5 v).
`vgdl=volt`	Reports the condition if Vgd is less than the specified lower bound voltage value.
`vgdu=volt`	Reports the condition if Vgd is greater than the specified upper bound voltage value.
`vgsl=volt`	Reports the condition if Vgs is less than the specified lower bound voltage value.
`vgsu=volt`	Reports the condition if Vgs is greater than the specified upper bound voltage value.
`vgbl=volt`	Reports the condition if Vgb is less than the specified lower bound voltage value.

`vgbu=volt`	Reports the condition if Vgb is greater than the specified upper bound voltage value.
`vdsl=volt`	Reports the condition if Vds is less than the specified lower bound voltage value.
`vdsu=volt`	Reports the condition if Vds is greater than the specified upper bound voltage value.
`vdbl=volt`	Reports the condition if Vdb is less than the specified lower bound voltage value.
`vdbu=volt`	Reports the condition if Vdb is greater than the specified upper bound voltage value.
`vsbl=volt`	Reports the condition if Vsb is less than the specified lower bound voltage value.
`vsbu=volt`	Reports the condition if Vsb is greater than the specified upper bound voltage value
`cond=expression`	Defines the conditional expression as the checking criteria. When the condition is met, the simulator generates a report. The conditional expression supports the following operators: `<`, `>`, `<=`, `>=`, `==`, `||`, `&&`, and variables: *vgs*, *vgd*, *vgb*, *vds*, *vdb*, *vsb*, *l*, *w*.
`num=n`	Specifies the maximum number of warnings generated by a particular check command (default value is 300 warnings).
`rpt_path=0|1`	Specifies whether or not to report the conducting paths. If set to `0`, no paths are reported (default). If set to `1`, the conducting paths from the MOSFET terminals to the voltage sources are reported.
`rpt_node=0|1`	Specifies whether or not to report the node voltages propagated from voltage sources. If set to 0, the node voltage is not reported (default). If set to 1, both the minimum and maximum values of the node voltages propagated from voltage sources, and the depth of propagation paths, are reported.
`ppos_nneg=0|1`	When `ppos_nneg` is set to `1`, positive voltage sources can only be propagated through PMOSFETs, and negative or zero voltage sources can only be propagated through NMOSFETs (default value is 0; no limitation on the type of MOSFETs during voltage propagation).

`pwl_time=time`	When specified, the PWL voltage source is replaced by a constant voltage source for the propagation of voltage. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only voltage from constant voltage sources is propagated).
`vsrc=[elem_name vmin vmax ...]`	When specified, voltage from the voltage source element <`elem_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` for propagation to begin. If `vlth` is set, `vmin` needs to be lower than or equal to `vlth` for propagation to begin (by default, only voltage from constant voltage sources is propagated). Multiple vsources are supported by the simulator.
`vsrcnode=[node_name vmin vmax ...]`	When specified, voltage from the voltage source node <`node_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` for propagation to begin. If `vlth` is set, `vmin` needs to be lower than or equal to `vlth` for propagation to begin (by default, only voltage from constant voltage sources is propagated). Multiple vsource nodes are supported by the simulator.
`xt_vsrc=0|1`	When `xt_vsrc` is set to 1, voltage propagation starts only from highest and lowest constant voltage sources. When `ext_vsrc` is set to 0, voltage propagation starts from all constant voltage sources. In either case, the selection is subject to the rules set by `vhth` and `vlth` (default value is 0).

Example 1

In the following Spectre example

```
usim_report chk_mosv chk1 model=nch inst=[*] xinst=[x1.x2 x1.x3] vgsu=1.5 vdsl=0.1
rpt_path=1
```

The Virtuoso UltraSim simulator checks if the voltage for all of the `nch` MOSFETs Vgs and Vds are within the specified bounds. The transistors located in instances `x1.x2` and `x1.x3` are excluded from the voltage check. The MOSFETs with Vgs>1.5 or Vds<0.1 are reported in the *`xxxx`*`.rpt_chkmosv` log file. With `rpt_path=1`, the conducting path from the MOSFET terminals to voltage sources is reported.

Example 2

In the following SPICE example

```
.usim_report chk_mosv chk2 model=nch inst=[x1.*] skipinst=[x1.x2] vhth=0.9
vgsu=1.5
```

The simulator checks if the voltage of `nch` MOSFETs located in instance `x1` for Vgs is less than 1.5 v. The voltage propagation only starts from voltage sources with values equal to or greater than 0.9 v. Voltage is not propagated through instance `x1.x2`. The MOSFETs with Vgs>1.5 are reported.

Example 3

In the following Spectre example

```
usim_report chk_mosv chk3 model=nch subckt=[nor2 nand2] maxmos=2 ppos_nneg=1
vgsu=1.5
```

The simulator checks the voltage of `nch` MOSFETs belonging to subcircuit `nor2` or `nand2`. The MOSFET is reported if Vgs>1.5.

During voltage propagation, only the nodes that can be connected to a voltage source by going through a maximum of two MOSFETs are considered. With `ppos_nneg=1`, positive voltage sources can only be propagated through PMOSFETs, and negative or zero voltage sources can only be propagated through NMOSFETs.

Example 4

In the following SPICE example

```
.usim_report chk_mosv chk4 model=pch cond='vgs<1 || vds>1.8'
```

The simulator checks the voltage of all `nch` MOSFETs. If a `nch` MOSFET has Vgs<1 or Vds>1.8, then the MOSFET is reported in the *`xxxx`*`.chk_mosv` log file.

The following is an example of a *`xxxx`*`.chk_mosv` log file.

```
Total of 4 Warnings reported in mosv2.

 Title   Model   Subckt    Vd   Vg           Vs           Vb   Instance
 chk1    nmos    vco_50m   --   1.1000e+00   0.0000e+00   --   x1.mn1i1051
 chk1    nmos    vco_50m   --   1.1000e+00   0.0000e+00   --   x1.mn1i1055
 chk1    nmos    nor2      --   2.5000e+00   0.0000e+00   --   x1.x1i1023.mn1i1
 chk1    nmos    nor2      --   2.5000e+00   0.0000e+00   --   x1.x1i1023.mn1i7

Total of 2 Warnings reported in chk2.

 chk2    nmos    inv       2.5000e+00   --   0.0000e+00   --   x1.x1i1036.mn1i2
 chk2    nmos    inv       2.5000e+00   --   0.0000e+00   --   x1.x1i1037.mn1
```

Static NMOS and PMOS Bulk Forward-Bias Checks

Static NMOS Bulk Forward-Bias Check

Spectre Syntax

```
usim_report chk_nmosb title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vlth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

SPICE Syntax

```
.usim_report chk_nmosb title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vlth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

Description

This command is used to check if the bulk to drain/source junctions of NMOSFETs become forward-biased.

Note: Check is performed after the netlist file is parsed.

A warning message is generated when the bulk bias voltage meets following condition:

```
min(Vb)>=min (Vd, Vs) + <vt>
```

where `vt` is the p-n junction threshold voltage of the NMOSFETs being checked. The report file format is `*xxxx*.rpt_chknmosb`.

Arguments

`title`	Title of report.

`model=model_name`	Specifies the model to be checked. The voltage check is applied to transistors with model card name `model_name`.
`subckt`	Specifies the subcircuits to be checked. The voltage check is applied to transistors belonging to subcircuits. Wildcard (`*`) characters can be used.
`inst`	Specifies the instances to be checked. The voltage check is applied to transistors belonging to instances (wildcard characters can be used).
`xsubckt`	Specifies the subcircuits not to be checked (wildcard characters can be used).
`xinst`	Specifies the instances not to be checked (wildcard characters can be used).
`skipsubckt`	Specifies that elements inside designated subcircuits skip voltage propagation (wildcard characters can be used).
`skipinst`	Specifies that elements inside designated instances skip voltage propagation (wildcard characters can be used).
`maxmos=n`	Maximum number of MOSFETs in the voltage propagation path between the voltage source and the MOSFET terminals (default is infinity).
`vt=volt`	Threshold voltage for p-n junction of NMOSFETs being checked (default value of `vt` for NMOSFET is 0.3 v).
`vlth=volt`	Voltage propagation starts only from constant voltage sources with values less than or equal to `vlth` (default value is 0.4 v)
`vnth=volt`	NMOSFET threshold voltage. This value is used to calculate the voltage drop across a NMOSFET channel during voltage propagation (default value is 0.5 v).
`vpth=volt`	PMOSFET threshold voltage. This value is used to calculate the voltage drop across a PMOSFET channel during voltage propagation (default value is -0.5 v).
`num=n`	Specifies the maximum number of warnings generated by a particular check command (default value is 300 warnings).
`rpt_path=0|1`	Specifies whether or not to report the conducting paths. If set to 0, no paths are reported (default). If set to 1, the conducting paths from the MOSFET terminals to the voltage sources are reported.

`rpt_node=0|1`
: Specifies whether or not to report the node voltages propagated from voltage sources. If set to 0, the node voltage is not reported (default). If set to 1, both the minimum and maximum values of the node voltages propagated from voltage sources, and the depth of propagation paths, are reported.

`pwl_time=time`
: When specified, the PWL voltage source is replaced by a constant voltage source for the propagation of voltage. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only voltage from constant voltage sources is propagated).

`vsrc=[elem_name vmin vmax ...]`
: When specified, voltage from the voltage source element <`elem_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` has to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsources are supported.

`vsrcnode=[node_name vmin vmax ...]`
: When specified, voltage from the voltage source node <`node_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` has to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsource nodes are supported.

`xt_vsrc=0|1`
: When `xt_vsrc` is set to 1, voltage propagation starts only from highest and lowest constant voltage sources. When `ext_vsrc` is set to 0, voltage propagation starts from all of the constant voltage sources. In either case, the selection is subject to the rules set by `vhth` and `vlth` (default value is 0).

Example 1

In the following Spectre example

```
usim_report chk_nmosb chk1 model=nch inst=[*] xinst=[x1.x2 x1.x3] vt=0.5
rpt_path=1
```

The Virtuoso UltraSim simulator checks if all of the `nch` NMOSFETs bulk to drain/source junctions become forward-biased. The threshold voltage is 0.5 v. The transistors located in instances `x1.x2` and `x1.x3` are excluded from the bulk forward-bias check. The NMOSFETs with bulk forward-bias are reported. With `rpt_path=1`, the conducting path from the MOSFET terminals to voltage sources is also reported.

Example 2

In the following SPICE example

```
.usim_report chk_nmosb chk2 model=nch inst=[x1.*] skipinst=[x1.x2] vhth=0.9
maxmos=2
```

The simulator checks if `nch` NMOSFETs located in instance `x1` for bulk to drain/source junctions become forward-biased. The voltage propagation starts only from voltage sources with values equal to or greater than 0.9 v. Voltage is not propagated through instance `x1.x2`. Only the nodes that can be connected to a voltage source by going through a maximum of two MOSFETs are considered. After the check is complete, NMOSFETs with bulk forward-bias are reported.

Static PMOS Bulk Forward-Bias Check

Spectre Syntax

```
usim_report chk_pmosb title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vhth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

Spectre Syntax

```
.usim_report chk_pmosb title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vhth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

Description

This command is used to check if the bulk to drain/source junctions of PMOSFETs become forward-biased.

Note: Check is performed after the netlist file is parsed.

A warning message is generated when the bulk bias voltage meets the following condition:

```
max(Vb)<=max (Vd, Vs) + <vt>
```

where `vt` is the p-n junction threshold voltage of the PMOSFETs being checked. The report file format is *`xxxx`*`.rpt_chkpmosb`.

Note: The `chk_pmosb` command is only supported in the new Virtuoso UltraSim simulation front end (SFE) parser. Use the `+csfe` command line option to enable SFE.

Arguments

`title`	Title of report.
`model=model_name`	Specifies the model to be checked. The bias check is applied to transistors with model card name `model_name`.
`subckt`	Specifies the subcircuits to be checked. The voltage check is applied to transistors belonging to subcircuits. Wildcard (`*`) characters can be used.
`inst`	Specifies the instances to be checked. The voltage check is applied to transistors belonging to instances (wildcard characters can be used).
`xsubckt`	Specifies the subcircuits not to be checked (wildcard characters can be used).
`xinst`	Specifies the instances not to be checked (wildcard characters can be used).
`skipsubckt`	Specifies that elements inside designated subcircuits skip voltage propagation (wildcard characters can be used).
`skipinst`	Specifies that elements inside designated instances skip voltage propagation (wildcard characters can be used).
`maxmos=n`	Maximum number of MOSFETs in the voltage propagation path between the voltage source and the MOSFET terminals.
`vt=volt`	Threshold voltage for p-n junction of PMOSFETs being checked (default value of `vt` for PMOSFET is -0.3 v).
`vhth=volt`	Voltage propagation starts only from constant voltage sources with values greater than or equal to `vhth` (default value is 0.7 v).
`vnth=volt`	NMOSFET threshold voltage. This value is used to calculate the voltage drop across a NMOSFET channel during voltage propagation (default value is 0.5 v).

`vpth=volt`	PMOSFET threshold voltage. This value is used to calculate the voltage drop across a PMOSFET channel during voltage propagation (default value is -0.5 v).
`num=n`	Specifies the maximum number of warnings generated by a particular check command (default value is 300 warnings).
`rpt_path=0|1`	Specifies whether or not to report the conducting paths. If set to 0, no paths are reported (default). If set to `1`, the conducting paths from the MOSFET terminals to the voltage sources are reported.
`rpt_node=0|1`	Specifies whether or not to report the node voltages propagated from voltage sources. If set to 0, the node voltage is not reported (default). If set to 1, both the minimum and maximum values of the node voltages propagated from voltage sources, and the depth of propagation paths, are reported.
`pwl_time=time`	When specified, the PWL voltage source is replaced by a constant voltage source for the propagation of voltage. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only voltage from constant voltage sources is propagated).
`vsrc=[elem_name vmin vmax ...]`	When specified, voltage from the voltage source element <`elem_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` needs to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsources are supported.
`vsrcnode=[node_name vmin vmax ...]`	When specified, voltage from the voltage source node <`node_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` needs to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsource nodes are supported.
`xt_vsrc=0|1`	When `xt_vsrc` is set to 1, voltage propagation starts only from highest and lowest constant voltage sources. When `ext_vsrc` is set to 0, voltage propagation starts from all of the constant voltage sources. In either case, the selection is subject to the rules set by `vhth` and `vlth` (default value is 0).

Example 1

In the following Spectre example

```
usim_report chk_pmosb chk1 model=pch inst=[*] xinst=[x1.x2 x1.x3] vt=-0.5
rpt_path=1
```

The Virtuoso UltraSim simulator checks if all of the `pch` PMOSFETs bulk to drain/source junctions become forward-biased. The threshold voltage is -0.5 v. The transistors located in instances `x1.x2` and `x1.x3` are excluded from the bulk forward-bias check. The PMOSFETs with bulk forward-bias are reported. With `rpt_path=1`, the conducting path from the MOSFET terminals to voltage sources is also reported.

Example 2

In the following SPICE example

```
.usim_report chk_pmosb chk2 model=pch inst=[x1.*] skipinst=[x1.x2] vhth=0.9
maxmos=2
```

The simulator checks if `pch` PMOSFETs located in instance `x1` for bulk to drain/source junctions become forward-biased. The voltage propagation starts only from voltage sources with values equal to or greater than 0.9 v. Voltage is not propagated through instance `x1.x2`. Only the nodes that can be connected to a voltage source by going through a maximum of two MOSFETs are considered. After checking, PMOSFETs with bulk forward-bias are reported.

Detect Conducting NMOSFETs and PMOSFETs

Detect Conducting NMOSFETs

Spectre Syntax

```
usim_report chk_nmosvgs title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vlth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

SPICE Syntax

```
.usim_report chk_nmosvgs title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vlth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

Description

This command allows you to check MOSFETs with n-type channels (NMOSFETs) to detect whether the minimum gate voltage is greater than the minimum drain/source voltages.

Note: Check is performed after the netlist file is parsed.

A warning message is generated when the gate voltage meets the following condition:

```
min(Vg)>=min(Vd, Vs) + <vt>
```

where `vt` is the p-n junction threshold voltage of the NMOSFETs being checked. The report format is *`xxxx`*`.rpt_chknmosvgs`.

Arguments

`title`	Title of report.

`model=model_name`	Specifies the model to be checked. The bias check is applied to transistors with model card name `model_name`.
`subckt`	Specifies the subcircuits to be checked. The voltage check is applied to transistors belonging to subcircuits. Wildcard (`*`) characters can be used.
`inst`	Specifies the instances to be checked. The voltage check is applied to transistors belonging to instances (wildcard characters can be used).
`xsubckt`	Specifies the subcircuits not to be checked (wildcard characters can be used).
`xinst`	Specifies the instances not to be checked (wildcard characters can be used).
`skipsubckt`	Specifies that elements inside designated subcircuits skip voltage propagation (wildcard characters can be used).
`skipinst`	Specifies that elements inside designated instances skip voltage propagation (wildcard characters can be used).
`maxmos=n`	Maximum number of MOSFETs in the voltage propagation path between the voltage source and the MOSFET terminals.
`vt=volt`	Threshold voltage for p-n junction of NMOSFETs being checked (default value of `vt` for NMOSFET is 0.3 v).
`vlth=volt`	Voltage propagation starts only from constant voltage sources with values less than or equal to `vlth` (default value is 0.4 v).
`vnth=volt`	NMOSFET threshold voltage. This value is used to calculate the voltage drop across a NMOSFET channel during voltage propagation (default value is 0.5 v).
`vpth=volt`	PMOSFET threshold voltage. This value is used to calculate the voltage drop across a PMOSFET channel during voltage propagation (default value is -0.5 v).
`num=n`	Specifies the maximum number of warnings generated by a particular check command (default value is 300 warnings).
`rpt_path=0|1`	Specifies whether or not to report the conducting paths. If set to 0, no paths are reported (default). If set to 1, the conducting paths from the MOSFET terminals to the voltage sources are reported.

`rpt_node=0|1`	Specifies whether or not to report the node voltages propagated from voltage sources. If set to 0, the node voltage is not reported (default). If set to 1, both the minimum and maximum values of the node voltages propagated from voltage sources, and the depth of propagation paths, are reported.
`pwl_time=time`	When specified, the PWL voltage source is replaced by a constant voltage source for the propagation of voltage. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only voltage from constant voltage sources is propagated).
`vsrc=[elem_name vmin vmax ...]`	When specified, voltage from the voltage source element <`elem_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` needs to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsources are supported.
`vsrcnode=[node_name vmin vmax ...]`	When specified, voltage from the voltage source node <`node_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` needs to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsource nodes are supported.
`xt_vsrc=0|1`	When `xt_vsrc` is set to 1, voltage propagation starts only from highest and lowest constant voltage sources. When `ext_vsrc` is set to 0, voltage propagation starts from all of the constant voltage sources. In either case, the selection is subject to the rules set by `vhth` and `vlth` (default value is 0).

Example 1

In the following Spectre example

```
usim_report chk_nmosvgs chk1 model=nch inst=[*] xinst=[x1.x2 x1.x3] vt=0.5
rpt_path=1
```

The Virtuoso UltraSim simulator checks all of the `nch` NMOSFETs to determine if they are conducting. The threshold voltage is -0.5 v. The transistors located in instances `x1.x2` and `x1.x3` are excluded from the check. NMOSFETs with a minimum gate voltage greater than the minimum drain/source voltages are reported. With `rpt_path=1`, the conducting path from the MOSFET terminals to voltage sources is also reported.

Example 2

In the following SPICE example

```
.usim_report chk_nmosb chk2 model=nch inst=[x1.*] skipinst=[x1.x2] vhth=0.9
maxmos=2
```

The simulator checks the `nch` NMOSFETs located in instance `x1` to determine if they are conducting. The voltage propagation starts only from voltage sources with values equal to or greater than 0.9 v. Voltage is not propagated through instance `x1.x2`. Only the nodes that can be connected to a voltage source by going through a maximum of two MOSFETs are considered. After checking, NMOSFETs with a minimum gate voltage greater than the minimum drain/source voltages are reported.

Detect Conducting PMOSFETs

Spectre Syntax

```
usim_report chk_pmosvgs title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vhth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

SPICE Syntax

```
.usim_report chk_pmosvgs title <model=model_name> <subckt=[subckt1 subckt2 …]>
     <inst=[inst1 inst2 …]> <xsubckt=[xsubckt1 xsubckt2 …]> <xinst=[xinst1 xinst2
     …]> <skipsubckt=[skipsubckt1 skipsubckt2 …]> <skipinst=[skipinst1 skipinst2
     …]> <maxmos=n> <vt=volt> <vhth=volt> <vnth=volt> <vpth=volt> <num=n>
     <rpt_path=0|1> <rpt_node=0|1> <pwl_time=time> <vsrc=[elem1 vmin1 vmax1
     elem2 vmin2 vmax2 …]> <vsrcnode=[node1 vmin1 vmax1 node2 vmin2 vmax2 …]>
     <xt_vsrc=0|1>
```

Description

This command allows you to check MOSFETs with p-type channels (PMOSFETs) to detect whether the maximum gate voltage is less than the maximum drain/source voltages without running a transient simulation.

Note: Check is performed after the netlist file is parsed.

A warning message is generated when the gate voltage meets the following condition:

```
max(Vg)<=max(Vd, Vs) + <vt>
```

where `vt` is the p-n junction threshold voltage of the PMOSFETs being checked. The report format is *`xxxx`*`.rpt_chkpmosvgs`.

Arguments

`title`	Title of report.
`model=model_name`	Specifies the model to be checked.The bias check is applied to transistors with model card name `model_name`.
`subckt`	Specifies the subcircuits to be checked. The voltage check is applied to transistors belonging to subcircuits. Wildcard (`*`) characters can be used.
`inst`	Specifies the instances to be checked. The voltage check is applied to transistors belonging to instances (wildcard characters can be used).
`xsubckt`	Specifies the subcircuits not to be checked (wildcard characters can be used).
`xinst`	Specifies the instances not to be checked (wildcard characters can be used).
`skipsubckt`	Specifies that elements inside designated subcircuits skip voltage propagation (wildcard characters can be used).
`skipinst`	Specifies that elements inside designated instances skip voltage propagation (wildcard characters can be used).
`maxmos=n`	Maximum number of MOSFETs in the voltage propagation path between the voltage source and the MOSFET terminals.
`vt=volt`	Threshold voltage for p-n junction of PMOSFETs being checked (default value of `vt` for PMOSFET is -0.3 v).
`vhth=volt`	Voltage propagation starts only from constant voltage sources with values greater than or equal to `vhth` (default value is 0.7 v)
`vnth=volt`	NMOSFET threshold voltage. This value is used to calculate the voltage drop across a NMOSFET channel during voltage propagation (default value is 0.5 v).
`vpth=volt`	PMOSFET threshold voltage. This value is used to calculate the voltage drop across a PMOSFET channel during voltage propagation (default value is -0.5 v).

`num=n`	Specifies the maximum number of warnings generated by a particular check command (default value is 300 warnings).
`rpt_path=0|1`	Specifies whether or not to report the conducting paths. If set to 0, no paths are reported (default). If set to 1, the conducting paths from the MOSFET terminals to the voltage sources are reported.
`rpt_node=0|1`	Specifies whether or not to report the node voltages propagated from voltage sources. If set to 0, the node voltage is not reported (default). If set to 1, both the minimum and maximum values of the node voltages propagated from voltage sources, and the depth of propagation paths, are reported.
`pwl_time=time`	When specified, the PWL voltage source is replaced by a constant voltage source for the propagation of voltage. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only voltage from constant voltage sources is propagated).
`vsrc=[elem_name vmin vmax ...]`	When specified, voltage from the voltage source element <`elem_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` needs to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsources are supported.
`vsrcnode=[node_name vmin vmax ...]`	When specified, voltage from the voltage source node <`node_name`> is propagated with values `vmin` and `vmax`. If `vhth` is set, `vmax` needs to be greater than or equal to `vhth` to start propagation. If `vlth` is set, `vmin` needs to be less than or equal to `vlth` to start propagation (by default, only voltage from constant voltage sources is propagated). Multiple vsource nodes are supported.
`xt_vsrc=0|1`	When `xt_vsrc` is set to 1, voltage propagation starts only from highest and lowest constant voltage sources. When `ext_vsrc` is set to 0, voltage propagation starts from all of the constant voltage sources. In either case, the selection is subject to the rules set by `vhth` and `vlth` (default value is 0).

Example 1

In the following Spectre example

```
usim_report chk_pmosvgs chk1 model=pch inst=[*] xinst=[x1.x2 x1.x3] vt=0.5
rpt_path=1
```

The Virtuoso UltraSim simulator checks all of the `pch` PMOSFETs to determine if they are conducting. The threshold voltage is 0.5 v. The transistors located in instances `x1.x2` and `x1.x3` are excluded from the check. PMOSFETs with a maximum gate voltage less than the maximum drain/source voltages are reported. With `rpt_path=1`, the conducting path from the MOSFET terminals to voltage sources is also reported.

Example 2

In the following SPICE example

```
.usim_report chk_pmosb chk2 model=pch inst=[x1.*] skipinst=[x1.x2] vhth=0.9
maxmos=2
```

The simulator checks the `pch` PMOSFETs located in instance `x1` to determine if they are conducting. The voltage propagation starts only from voltage sources with values equal to or greater than 0.9 v. Voltage is not propagated through instance `x1.x2`. Only the nodes that can be connected to a voltage source by going through a maximum of two MOSFETs are considered. After checking, PMOSFETs with a maximum gate voltage less than the maximum drain/source voltages are reported.

Detect NMOSFETs Connected to VDD

Spectre Syntax

```
usim_report chk_nmos2vdd title <model=model_name> <subckt=subckt_name>
     <inst=inst_name> <xsubckt=subckt_name> <xinst=inst_name> <vhth=volt>
     <node=[all | gate | drain | source | bulk] > <num=n> <pwl_time=time>
```

SPICE Syntax

```
.usim_report chk_nmos2vdd title <model=model_name> <subckt=subckt_name>
     <inst=inst_name> <xsubckt=subckt_name> <xinst=inst_name> <vhth=volt>
     <node=[all | gate | drain | source | bulk] > <num=n> <pwl_time=time>
```

Description

This command allows you to detect NMOSFETs with terminal(s) that are directly connected to the constant or PWL voltage sources, which have a voltage value higher than `vhth` (without running transient simulation). When you run this command, the software generates a report file named as `xxxx.rpt_chknmos2vdd`.

Important

When you use this command:

- Inductors and resistors less than 1M ohm are considered as short.
- Diodes are considered as open.

Arguments

`title`	Title of report.
`model=model_name`	Specifies the MOSFET model to be checked.
`subckt=subckt_name`	Specifies the subcircuits to be checked. Wildcard (`*`) characters can be used.
`inst=inst_name`	Specifies the instances to be checked. Wildcard (*) characters can be used.
`xsubckt=subckt_name`	Specifies the subcircuits that should not be checked. Wildcard (*) characters can be used.
`xinst=inst_name`	Specifies the instances that should not be checked. Wildcard (*) characters can be used.
`vhth=volt`	Specifies the threshold value of constant voltage source to be checked. The default value is 0.7V.
`node=[terminal_name ...]`	Specifies the terminal names to be checked for connection to voltage sources. The terminal names can be `all` or a combination of `drain`, `source`, `gate`, and `bulk`. When `all` is specified, all the terminals except `gate` will be checked. The default value is `all`.
`num=n`	Specifies the maximum number of warnings generated by the command. The default value is 300.
`pwl_time=time`	Replaces the PWL voltage source with a constant voltage source for checking. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only the constant voltage sources are checked).

Example 1

In the following Spectre example

```
usim_report chk_nmos2vdd chk1 model=n2 node = [drain source]
```

The Virtuoso UltraSim simulator checks if any NMOSFETs of model `n2` have the drain and/or source terminal connected to a voltage source with value higher than 0.7V.

Example 2

In the following SPICE example

```
.usim_report chk_nmos2vdd  chk2   vhth=1
```

The Virtuoso UltraSim simulator checks if any NMOSFETs have the drain, source, or bulk terminal connected to a voltage source with value higher than 1V.

Detect PMOSFETs Connected to GND

Spectre Syntax

```
usim_report chk_pmos2gnd title <model=model_name> <subckt=subckt_name>
     <inst=inst_name> <xsubckt=subckt_name> <xinst=inst_name> <vlth=volt>
     <node=[all | gate | drain | source | bulk] > <num=n> <pwl_time=time>
```

SPICE Syntax

```
.usim_report chk_pmos2gnd title <model=model_name> <subckt=subckt_name>
     <inst=inst_name> <xsubckt=subckt_name> <xinst=inst_name> <vlth=volt>
     <node=[all | gate | drain | source | bulk] > <num=n> <pwl_time=time>
```

Description

This command allows you to detect PMOSFETs with terminal(s) that are directly connected to the constant or PWL voltage sources, which have a voltage value lower than `vlth` (without running transient simulation). When you run this command, the software generates a report file named as `xxxx.rpt_chkpmos2gnd`.

Important

When you use this command:

- Inductors and resistors less than 1M ohm are considered as short.

- Diodes are considered as open.

Arguments

`title`	Title of report.
`model=model_name`	Specifies the MOSFET model to be checked.
`subckt=subckt_name`	Specifies the subcircuits to be checked. Wildcard (`*`) characters can be used.
`inst=inst_name`	Specifies the instances to be checked. Wildcard (*) characters can be used.
`xsubckt=subckt_name`	Specifies the subcircuits that should not be checked. Wildcard (*) characters can be used.
`xinst=inst_name`	Specifies the instances that should not be checked. Wildcard (*) characters can be used.
`vlth=volt`	Specifies the threshold value of constant voltage source to be checked. The default value is 0.4 v.
`node=[terminal_name ...]`	Specifies the terminal names to be checked for connection to voltage sources. The terminal names can be `all` or a combination of `drain`, `source`, `gate`, and `bulk`. When `all` is specified, all the terminals except `gate` will be checked. The default value is `all`.
`num=n`	Specifies the maximum number of warnings generated by the command. The default value is 300.

Example 1

In the following Spectre example

```
usim_report chk_pmos2gnd chk3 model=p2 node = [drain source]
```

The Virtuoso UltraSim simulator checks if any PMOSFETs of model `p2` have the drain and/or source terminal connected to voltage source with value lower than 0.4V.

Example 2

In the following SPICE example

```
.usim_report chk_pmos2vdd  chk24 inst=[x1 x2]
```

The Virtuoso UltraSim simulator checks if any PMOSFETs of the instances x1 and x2 have the drain, source, or bulk terminal connected to a voltage source with value lower than 0.4V.

Static Maximum Leakage Path Check

```
.usim_report chk_maxleak title <vnth=v> <vpth=v> <num=n>
```

Description

The `chk_maxleak` command is used to detect static DC leakage paths between all voltage sources. All reported maximum leakage paths are written into a file with the extension `rpt_maxleak`.

Arguments

`title`	Reports titles of warning messages
`vnth=volt`	NMOSFET threshold voltage (default value is 0 V)
`vpth=volt`	PMOSFET threshold voltage (default value is 0 V)
`num=n`	Number of paths reported

Example

```
.usim_report chk_maxleak checkleak vnth=0.5 vpth=-0.5
```

The Virtuoso UltraSim simulator detects all static maximum leakage paths between voltage sources and generates the following report:

```
.TITLE 'This file is :./test.rpt_maxleak'
Static Leakage Path Report For checkleak
Leakage Path From vdd! (1.80V) to 0 (0.00V)
   Element              Between Node            And Node
   Vvdd!                vdd!
   M3                   vdd!                    net034
   M5                   net034                  0
End Path
```

Static High Impedance Check

Spectre Syntax

```
usim_report chk_hznode title <vnth=volt> <vpth=volt> <fanout=value>
     <pwl_time=time> <num=n>
```

SPICE Syntax

```
.usim_report chk_hznode title <vnth=volt> <vpth=volt> <fanout=value>
     <pwl_time=time> <num=n>
```

Description

This command allows you to detect high impedance nodes without running DC or transient simulations, and generates a `.rpt_hznode` report (the entire circuit is checked). A node is in high impedance state if there is no possible conducting path between the node and any voltage source or ground.

The following rules apply in the connectivity evaluation:

- MOSFET and JFET of n-type are considered `on` if Vg-Vs >= Vnth
- MOSFET and JFET of p-type are considered `on` if Vs-Vg >= -Vpth
- Resistor larger than Rth (See the <u>Notes</u> section for the definiton of Rth) is considered as open.
- BJT, diode, resistor, and voltage sources are always considered `on`
- Capacitor and current sources are always assumed to be `off`

Arguments

`title`	Title of report.
`vnth=volt`	NMOSFET threshold voltage (default value is 0.5 V).
`vpth=volt`	PMOSFET threshold voltage (default value is -0.5 V).

`fanout=value`	Defines the type of nodes to be reported: ■ **0** - all Hi-Z nodes ■ **1** - only Hi-Z nodes connected to a MOSFET gate ■ **2** - only Hi-Z nodes connected to a MOSFET body ■ **3** - only Hi-Z nodes connected to a MOSFET gate or BJT base ■ **4** - only Hi-Z nodes connected to a MOSFET gate, but not to the gate of MOSFETs with the drain and source shorted ■ **5** - only Hi-Z nodes connected to a MOSFET gate or BJT base, but not to the gate of MOSFETs with the drain and source shorted, or the base of BJTs with the collector and emitter shorted
`pwl_time=time`	When specified, the PWL voltage source is replaced by a constant voltage source for the propagation of voltage. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only voltage from constant voltage sources is propagated).
`num=n`	Specifies maximum number of warnings reported.

Example

```
.usim_report chk_hznode title vnth=0.4 vpth=-0.4 fanout=4
```

The Virtuoso UltraSim simulator detects all high impedance nodes (that is, nodes with no possible path to voltage sources or ground) which are connected to a MOSFET gate or BJT base using the defined threshold voltages for NMOS and PMOS devices.

Notes

- The resistance threshold value Rth can be changed by the following option:

  ```
  .usim_opt res_open = value
  ```

  The default value of Rth is 100 Mohm.

Static ERC Check

Spectre Syntax

```
.usim_report erc title <powergatebulk=1>  <underbiasbulk=1> <hotwell=1>
     <floatgate=1|2|3|4> <floatbulk=1|2> <dangle=1|2> <low2highvdd=1>
     <high2lowvdd=1> <powershort=1> <vhth=volt> <vlth=volt> <rmax=res>
     <pwl_time=time>
```

SPICE Syntax

```
.usim_report erc title <powergatebulk=1>  <underbiasbulk=1> <hotwell=1>
     <floatgate=1|2|3|4> <floatbulk=1|2> <dangle=1|2> <low2highvdd=1>
     <high2lowvdd=1> <powershort=1> <vhth=volt> <vlth=volt> <rmax=res>
     <pwl_time=time>
```

Description

The static ERC check allows you to detect the following electrical design rule violations without running simulation:

- MOSFET power switch whose bulk is not hard-wired to power supply.
- MOSFET with forward biased junction.
- MOSFET with bulk not hard-wired to power supply.
- Unconnected MSOFET gate.
- Unconnected MOSFET bulk.
- Dangling node.
- MOSFET in high VDD domain driven by MOSFETs in low VDD domain.
- MOSFET in low VDD domain driven by MOSFETs in high VDD domain.
- MOSFET directly shorting VDD and GND.

The static ERC check can be invoked without running DC or transient simulation, and it generates a report file (`***.rpt_erc`) listing the details of the violations based on the specified arguments.

Arguments

`title`	Title of report
`powergatebulk`	Reports PMOS powergate with bulk not connected to VDD, or NMOS powergate with bulk not connected to GND.
`underbiasbulk`	Reports NMOSFET with bulk biased at voltage level higher than S/D, or PMOSFET with bulk biased at voltage level lower than S/D.
`hotwell`	Reports MOSFET with bulk not connected to VDD or GND.
`floatgate`	Reports unconnected MOSFET gate (1 = check all nodes, 2 = exclude top level nodes, 3 = exclude MOSCAP gates, 4 = exclude top level nodes and MOSCAP gates)
`floatbulk`	Reports unconnected MOSFET bulk (1 = check all nodes, 2 = exclude top level nodes)
`dangle`	Reports dangling nodes (1 = check all nodes, 2 = exclude top level nodes).
`low2highvdd`	Reports MOSFETs in high VDD domain driven by MOSFETs in low VDD domain.
`high2lowvdd`	Reports MOSFETs in low VDD domain driven by MOSFETs in high VDD domain.
`powershort`	Reports MOSFETs with channel connected directly between VDD and GND.
`vhth=vol`	Any voltages above vhth are considered as VDD (default = 0.5v).
`vlth=vol`	Any voltages below vlth are considered as GND (default = 0.5v)
`rmax=res`	Maximum resistance values where a node is still considered connected to voltage source node. (default = 100Mohm)
`pwl_time=t`	If specified, pwl sources are considered same as dc source. The voltage level at time=t is used.

Examples

```
.usim_report  erc check_pwrgate  powergatebulk=1
```

Reports all powergates whose bulks are not connected to VDD or GND.

```
.usim_report erc check_pershort  powershort =1  vhth=2.0
```

Reports any MOSFET shorting VDD or GND. VDD is any source with voltage level higher than 2V.

```
.usim_report  erc check_floatgate floatgate=4
```

Reports all MOSFET floating gates but excludes top-level nodes and MOS capacitors.

Static DC Path Check

Spectre Syntax

```
usim_report chk_dcpath title  <vnth=volt> <vpth=volt> <pwl_time=time>
```

SPICE Syntax

```
.usim_report chk_dcpath title  <vnth=volt> <vpth=volt> <pwl_time=time>
```

Description

This command allows you to detect a DC path between voltage sources without running DC or transient simulation, and generates a report named `xxxx.rpt_dcpath`. The DC path can consist of MOSFETs, BJTs, diodes, inductors, and resistors with value less than 1G ohm. All other elements are treated as open.

When you run this command:

- NPN transistors are considered as conducting if VBE is greater than 0.5V.
- PNP transistors are considered as conducting if VBE is less than 0.5V.
- Diode is considered as conducting when it is forward biased by more than 0.5V.

Note: The entire circuit is checked.

Arguments

`title`	Title of report.
`vnth=volt`	Specifies the NMOSFET threshold voltage. The default value is 0.5 V.
`vpth=volt`	Specifies the PMOSFET threshold voltage. The default value is -0.5 V.
`num=n`	Specifies the maximum number of warnings generated by the command. The default value is 300.
`pwl_time=time`	Replaces the PWL voltage source with a constant voltage source for checking. The constant voltage value is equal to the PWL voltage source value at the specified time (by default, only the constant voltage sources are checked).

`rpt_path=0|1`	Specifies whether to report the conducting paths. If set to `0`, no paths are reported (default). If set to `1`, the conducting paths from the MOSFET terminals to the voltage sources are reported.

Example 1

In the following SPICE example

```
.usim_report chk_dcpath checkdcpath  rpt_path=1
```

The Virtuoso UltraSim simulator will detect DC paths between all voltage sources and report the list of elements in the conduction paths.

info Analysis

`infoname info what=… where=… extremes=..`

Description

This statement is similar to the Virtuoso Spectre® `info` statement and allows you to access input/output values and operating-point parameters. The parameter types include:

- Input parameters

  Input parameters are those you specify in the netlist file, such as the given length of a MOSFET or the saturation current of a bipolar resistor.

- Output parameters

  Output parameters are those the simulator computes, such as temperature-dependent parameters and the effective length of a MOSFET after scaling.

- Operating-point parameters

  Operating-point parameters are those that depend on the operating point.

Note: You need to specify the info analysis within the Spectre section of the netlist file.

You can also list the minimum and maximum values for the input, output, and operating-point parameters, along with the names of the components that have those values.

Arguments

`what=...`	Supports the following values: `inst`, `input`, `output`, `all`, `oppoint`, `models`
`where=...`	Supports the following values: `screen`, `nowhere`, `file`, `logfile`, and `rawfile`
`extremes=...`	Supports the following values: `yes`, `no`, and `only`

Saving Parameters

You specify parameters you want to save with the `info` statement `what` parameter. You can assign the following settings to this parameter:

Parameters	Descriptions
`inst`	Lists input parameters for instances of all components
`models`	Lists input parameters for models of all components
`input`	Lists input parameters for instances and models of all components
`output`	Lists effective and temperature-dependent parameter values
`all`	Lists input and output parameter values
`oppoint`	Lists operating-point parameters

You can also generate a summary of maximum and minimum parameter values with the `extremes` option.

Specifying the Output Destination

You can choose among several output destination options for the parameters you list with the `info` statement. With the `info` statement where parameter, you can

- Display the parameters on a screen, in a file, or a log file.
- Send the parameters to the raw file

  Note: The Virtuoso UltraSim simulator only supports `psf` format.

When the `info` analysis is called from a transient analysis or used inside of a sweep, the name of the `info` analysis is appended by the parent analysis.

Examples

For example

```
TranAnalysis tran stop=30n infotimes=[1n 10n] infoname=opinfo
opinfo info what=oppoint where=rawfile
```

produces operating-point information for times 1 ns and 10 ns in the raw data file.

In the next example

```
Inparams info what=models where=screen extremes=only
```

tells the simulator to send the maximum and minimum input parameters for all models to a screen (the section for the `info` report is `InParams`).

Partition and Node Connectivity Analysis

The Virtuoso UltraSim simulator lets you perform partition and node connectivity analysis using `usim_report` commands. The information is reported in a `.part_rpt` file. For example, if the netlist filename is `circuit.sp`, then the report is named `circuit.part_rpt`.

The `usim_report` commands are useful for debugging simulations. For example, checking the size of partitions and their activities, as well as checking node activity to verify bus nodes.

Partition Reports

Size

Spectre Syntax

```
usim_report partition type=size
```

SPICE Syntax

```
.usim_report partition type=size
```

Description

Reports the partition information for the 10 largest partitions and includes:

- Partition index and all of its instances
- Node information for each of the instances, including input ports, output ports, and internal nodes
- Element information for each of the instances

Example

In the following Spectre syntax example

```
usim_report partition type=size
```

tells the Virtuoso UltraSim simulator to report the partition information for the 10 largest partitions in a `.part_rpt` file.

Activity

Spectre Syntax

```
usim_report partition type=act
```

SPICE Syntax

```
.usim_report partition type=act
```

Description

Reports the partition information for the 10 most active partitions (same format as the partition size report). Also reports some of the activity information for the partitions.

Example

In the following SPICE syntax example

```
.usim_report partition type=act
```

tells the UltraSim Virtuoso simulator to report the partition information for the 10 most active partitions in a `.part_rpt` file.

Node

Spectre Syntax

```
usim_report partition type=conn node=[node1 node2...]
```

SPICE Syntax

```
.usim_report partition type=conn node=[node1 node2...]
```

Description

Reports the partitions connected to the specified node (same format as the partition size report).

Arguments

`node_name`	The name of the node to be analyzed

Example

In the following Spectre example

```
usim_report partition type=conn node=d0<15>
```

tells the UltraSim Virtuoso simulator to report all partitions connected to node `d0<15>`.

Node Connectivity Report

Spectre Syntax

```
usim_report node elem_threshold=num full_hiername=yes|no
```

SPICE Syntax

```
.usim_report node elem_threshold=num full_hiername=yes|no
```

Description

Reports nodes with an element connection larger than `elem_threshold` (default value for `elem_threshold` is 10). The report includes the following information for each node:

- Number of active devices connected to the node
- Number of channel connected devices for the node

Arguments

`elem_threshold`	Minimum number of node connections to be printed.
`full_hiername=yes|no`	Flag used to print the full hierarchical name for the reported nodes. ■ **`yes`** – prints the full hierarchical name. ■ **`no`** – prints the hierarchical name (default). For some types of circuits with complex hierarchies, the Virtuoso UltraSim simulator will print a limited hierarchy (starting from a specified subcircuit) to avoid generating a large report file. You can use `full_hiername=yes` to force the simulator to print the full hierarchical name.

Example

Spectre Syntax:

```
usim_report node elem_threshold=100
```

SPICE Syntax:

```
.usim_report node elem_threshold=100
```

tells the Virtuoso UltraSim simulator to report all nodes connected to more than 100 elements.

Warning Message Limit Categories

The Virtuoso UltraSim simulator allows you to customize how warning messages are handled by the simulator. The number of messages per warning category can be limited globally for all warnings (`usim_opt warning_limit`) or individually for each category (`usim_report warning_limit`). When the specified category limit is reached, the simulator notifies you that the warning messages are no longer being displayed. Dangling and floating node warnings are controlled by the number of reported nodes.

Description

Spectre Syntax

```
usim_report warning_limit=value warning_id=[id1 id2 ....]
```

SPICE Syntax

```
.usim_report warning_limit=value warning_id=[id1 id2 ....]
```

Defines the maximum number of warning messages printed for category IDs `id1` and `id2`. This option needs to be defined at the beginning of the netlist file in order to have an effect on all of the warning messages for the specified categories.

For more information about the key Virtuoso UltraSim simulator warning messages, refer to Table 8-1 on page 518.

Arguments

Table 8-1 Warning Limit Options

Option	Description
`warning_limit=`*`value`*	Number of warnings (integer, unitless; default is 5)
`id1`, `id2`	Warning limit applies to these warning message category IDs. **Note:** The prefix (component name) needs to be specified for the category ID.

Example

Spectre Syntax:

```
usim_report warning_limit = 20 warning_id=[USIM-1223 USIM-4003]
```

SPICE Syntax:

```
.usim_report warning_limit=20 warning_id=[USIM-1223 USIM-4003]
```

tells the simulator to print out 20 warning messages for `WARNING USIM-1223` and `USIM-4003`.

9

Netlist-Based EM/IR Flow

The Virtuoso® UltraSim™ netlist-based EM/IR flow packages electromigration (EM) and IR drop analysis capabilities into the Virtuoso UltraSim simulator. This flow is based on detailed standard parasitic format (DSPF) or standard parasitic exchange format (SPEF), and therefore is independent of the extraction tool.

The EM/IR flow uses the Virtuoso UltraSim hierarchical stitching technique to provide high capacity EM and IR analysis for large circuit designs, and is integrated into the Cadence Virtuoso/DFII environment platform. This flow also provides graphical display of EM and IR analysis results in colored, graphical violation maps along with textual report files. The graphical and textual outputs can be cross probed for debugging purposes, and the violation maps can be overlaid on top of the original layout view.

The EM/IR flow includes the following key elements,

When simulating data, the Virtuoso UltraSim simulator saves the intermediate binary database to disk. The binary database is converted into either the CDB or OpenAccess (OA) database for viewing as violation maps in the Virtuoso layout editor. The usimEmirUtil Tool is used to control the database conversion.

The EM/IR flow requires the following inputs,

- Pre-layout netlist file
- DSPF/SPEF files with the necessary geometry information for current density computation and violation map reconstruction (for example, coordinates of the parasitic resistors, and length and width of the resistors)

- Current density limit for which the file syntax is the same as the Virtuoso Analog ElectronStorm Option (VAEO) product syntax
- Original layout design in DFII database to overlay the violation maps on top of the layout view

  Note: The resistors are assumed to be rectangles and the violation maps are only approximations of the original layout design.

Simulating Data and Saving Files

Block-Level Solution

Spectre Syntax

```
usim_emir [type=all|selected] [nets=net1 net2….] format=[layout]
     file=”control_file” [start=start_time1] [stop=stop_time1]

usim_emir [type=all|selected] [nets=net1 net2….] format=[layout]
     file=”control_file” [start=start_time2] [stop=stop_time2]
```

SPICE Syntax

```
.usim_emir [type=all|selected] [nets=net1 net2….] format=[layout]
     file=”control_file” [start=start_time1] [stop=stop_time1]

.usim_emir [type=all|selected] [nets=net1 net2….] format=[layout]
     file=”control_file” [start=start_time2] [stop=stop_time2]
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.usim_emir`).

Description

To enable the Virtuoso UltraSim simulator to save the intermediate data files, you need to specify the `usim_emir` statement in the netlist file. This statement tells the simulator to save layout physical and voltage/current information for specified nets and associated resistors.

Arguments

type	The Virtuoso UltraSim simulator considers all or selected nets and resistors (default is all). If selected is used, you must specify all of the nets.
nets	Specifies the nets for which the necessary information is saved in the database. **Note:** Nets are applicable only when type=selected.
format	Specifies the layout design to be used in the netlist-based EM/IR flow. The simulator saves the physical and voltage/current information in a binary database for specified nets in a binary database (default is layout). **Note:** The format argument can also be used in conjunction with vavo (see Chapter 16, "VST/VAVO/VAEO Interfaces" for more information about the vavo argument).
file	Specifies the control file. Some commands (for example, geounit) in the control file require special parsing. Specifying file ensures that these commands take effect. See "Control File" on page 543 for more information about control file syntax.
start_time/ stop_time	Specifies the time window start and stop times. The start time default is the beginning of the transient simulation and the stop time default is the end of the transient simulation. Multiple time windows are supported and must be specified by different .usim_emir statements. **Note:** For the Virtuoso UltraSim/VAVO flow, the command used is usim_emir format=[vavo] (see Chapter 16, "VST/VAVO/VAEO Interfaces" for more information).

Intermediate binary files are saved to the database after the simulation ends. Using design.sp as a sample netlist file, the intermediate binary file naming convention is as follows:

- Design.emir0_bin
- Design_phys.data
- Design_phys.field
- Design_phys.layer

- `Design_phys.name`

Advanced EMIR Flow for Big Blocks and Full Chip

The advanced EMIR flow provides improved capability over the block-level solution described in the previous section. The advanced flow targets big blocks and full-chip EMIR analysis. In the advanced flow, two simulations are needed and as a result Ultrasim must be invoked twice, manually. The first simulation generates a file with a `.emirtap.sp` suffix. All the required options for performing the second simulation are set in the `*.emirtap.sp` file, and you need to run UltraSim for the second simulation using this `*.emirtap.sp` file.

Note: Because the netlist generated from the first round of simulation is the only input required for the second round of simulation, you need to specify the simulation option settings for only the first simulation.

Spectre Syntax

```
usim_emir  <iteration=value>  <tstep=value>  <time_window=[start1 stop1 start2
     stop2…]>

usim_emir type=[power|signal] nets=[netA <netB … >] <time_window=[start1 stop1
     start2 stop2…]>  file=filename
```

SPICE Syntax

```
.usim_emir  <iteration=value>  <tstep=value>  <time_window=[start1 stop1 start2
     stop2…]>

.usim_emir type=[power|signal] nets=[netA <netB … >] <time_window=[start1 stop1
     start2 stop2…]> file=filename
```

Note: Multiple `.usim_emir` statements are supported.

The advanced EMIR flow is enabled when the `type=power` or `type=signal` statements are used. When none of these statements are used, UltraSim uses the block-level solution.

Arguments

`usim_emir`	Indicates that EMIR analysis is needed for the specified nets.
`type=power`	Keyword `power` indicates that power net analysis will be performed.
`type=signal`	Keyword `signal` indicates that signal net analysis will be performed.

`nets` Specifies the nets for EMIR analysis. The complete hierarchical path of the nets must be specified. Wild cards are supported while specifying net names.

Note: The stitching option `spfskippwnet` is not required for stitching power nets in the advanced EMIR flow. For nets that are neither specified in power EMIR analysis statements nor signal EMIR analysis statements, the stitching is performed as per regular stitching options.

`time_window` Specifies the time window. Multiple time windows are supported. The default timing window is from the transient start time to the transient stop time. `time_window` can be specified in two ways:

- `time_window` (without `type=` specification) serves as global EMIR time window
- `time_window` with `type=` specification serves as local time window.

Note: Local time window specification takes higher priority.

`iteration` Specifies a positive integer value for the number of iterations to be performed. This option is available only for power net EMIR analysis. The expected behavior of different iteration numbers is as follows:

- 1: The signal nets are simulated first followed by power nets. Next, EMIR post-processing is performed.
- 2: After the last simulation in iteration=1, the final IR-drop information is applied to the signal nets, and UltraSim simulates the signal nets again. Next, UltraSim simulates the power nets again. Finally, EMIR post-processing is performed.

When the iteration number is set to more than 3, the above iterations continue until the desired iteration number is reached.

Each Simulation will need to be invoked manually. The default value for this option is 1.

`tstep` Specifies the time step used to save the tap current. Default is 20ps.

`file` Specifies the control file that will be used for the `usimEmirUtil` post-processing utility. The control file should be specified if the command `geounit`, which is specified in the control file, is needed.

See the *Control File* section for more information on `geounit`.

Examples

- Use the following settings if:
  - You have two power nets, VDD and VSS but only VDD needs EMIR analysis.
  - All the signal nets are stitched with C-only and VSS will not be stitched.
  - The time window is default.

```
.usim_emir type=power nets=[VDD]
.usim_opt spfrcnet=VDD
```

- Use the following settings if:
  - You have two power nets VDD and VSS but only VDD needs EMIR analysis.
  - VSS is not stitched.
  - All the signal nets are stitched with RC and the nets are subject to RC reduction.
  - Multiple time windows are needed.

```
.usim_emir type=power nets=[VDD] time_window=[0 10n 3u 4u]
.usim_opt postl=2 rshort=1 rvshort=1
```

- Use the following settings if you have two power nets VDD and VSS and you want to perform only signal EM analysis while power nets are not stitched:

```
.usim_emir type=signal nets=[*]
```

- Consider a situation where you are migrating from the block-level EMIR solution to the advanced EMIR solution, and you want to perform EMIR analysis on VDD power nets only. In addition, all the signals are C-only stitching.

  The option settings for block-level solution will be as follows:

```
.usim_opt spfrcnet=VDD
.usim_opt spfskippwnet=off
.usim_opt postl=0 rshort=0 rvshort=0
.usim_emir format=[layout]
.usim_pn node=VDD method=ups
```

  The corresponding option settings in the advanced EMIR flow will be as follows:

```
.usim_emir type=power nets=[VDD]
.usim_opt spfrcnet=VDD
.usim_pn node=VDD method=ups
```

- Consider a situation where you are migrating from the block-level EMIR solution to the advanced EMIR solution, and you want to perform EMIR analysis on VDD power nets only. In addition, all the signals are RC stitching.

The option settings for block-level solution will be as follows:

```
.usim_opt spfskippwnet=off
.usim_opt postl=2 rshort=2
.usim_opt postl=0 rshort=0 rvshort=0 scope=power
.usim_emir type=selected nets=[VDD] format=[layout]
.usim_pn node=VDD method=ups
```

The corresponding option settings in the advanced EMIR flow will be as follows:

```
.usim_emir type=power nets=[VDD]
.usim_opt postl=2 rshort=2
.usim_pn node=VDD method=ups
```

Mixed usage of block-level flow and the advanced EMIR flow is not supported. For example, consider that you are using the following option settings:

```
.usim_emir type=selected nets=[VDD VSS]
.usim_emir type=signal nets=[netA netB]
```

In this case, the command for the block-level solution will be disabled and the advanced EMIR analysis solution will take effect. In addition, a warning message will appear as shown:

```
WARNING (USIMDB-1311): The .usim_emir statement '.usim_emir type=selected start=0
stop=2e-08' has not nets or correct net type (power or signal). The UltraSim
simulator will ignore this statement. Check the .usim_emir statement where the nets
and/or type is defined before running the simulation again.
```

Running UltraSim with the `*.emirtap.sp` file generates the intermediate binary database for post-processing. For example, if you use `design.sp` as a netlist file, the naming convention for the intermediate binary files will be as follows:

```
design.emirtap.emir0_bin
design.emirtap_phys.data
design.emirtap_phys.field
design.emirtap_phys.layer
design.emirtap_phys.name
```

Simulations are the first step in EMIR flow. The generated binary database must be post-processed in order to generate EMIR violation maps and textual reports, which in turn will be displayed in Virtuoso. Post-processing is described in the following sections using the `design.emir0_bin` binary database generated from the block-level solution as an example. The post-processing for the advanced solution is the same as the block- level solution except that `design.emir0_bin` will need to be replaced with `design.emirtap.emir0_bin` (generated by the advanced EMIR solution).

Displaying Results for Analysis

To display the results of the simulation for analysis, perform the following procedure.

1. Assign the `_USIM2CDBCXTDIR` environment variable to the same location where the `usimemir.ini` and `usimemir.cxt` files reside.

   ```
   setenv _USIM2CDBCXTDIR $USIM_INSTALLATION_PATH/tools.<plat>/ultrasim/lib
   ```

2. Add the following information to your `.cdsinit` file.

   ```
   /* the following lines are needed to load usimemir cxt package */
   path=getShellEnvVar("_USIM2CDBCXTDIR")
   if(path then
       load(strcat(path "/usimemir.ini"))
       loadContext(strcat(path "/usimemir.cxt"))
   else
       hiGetAttention(0)
       hiDisplayAppDBox(
           ?name '_warning
           ?dboxBanner "Error"
           ?dboxText sprintf(nil "ERROR: Should set the environment
   _USIM2CDBCXTDIR before using usimemir package")
           ?buttonLayout 'Close
       )
   )
   ```

3. Set the proper environment variables for the DFII environment.

   For example,

   ```
   setenv CDSHOME /grid/cic/products/dfII/5.10.41_USR4/lnx86/red
   setenv MMSIMHOME /USIM_INSTALL/release
   setenv _USIM2CDBCXTDIR  ${MMSIMHOME}/tools.<plat>/ultrasim/lib
   set path = (${CDSHOME}/tools/bin ${CDSHOME}/tools/dfII/bin ${MMSIMHOME}/
   tools.lnx86/bin $path)
   setenv CDS_Netlisting_Mode Analog
   setenv LANG en_US
   setenv CDS_LOG_VERSION sequential
   setenv LD_LIBRARY_PATH ${MMSIMHOME}/tools/ultrasim/lib:${CDSHOME}/tools/
   lib:${LD_LIBRARY_PATH}
   ```

   Notes:

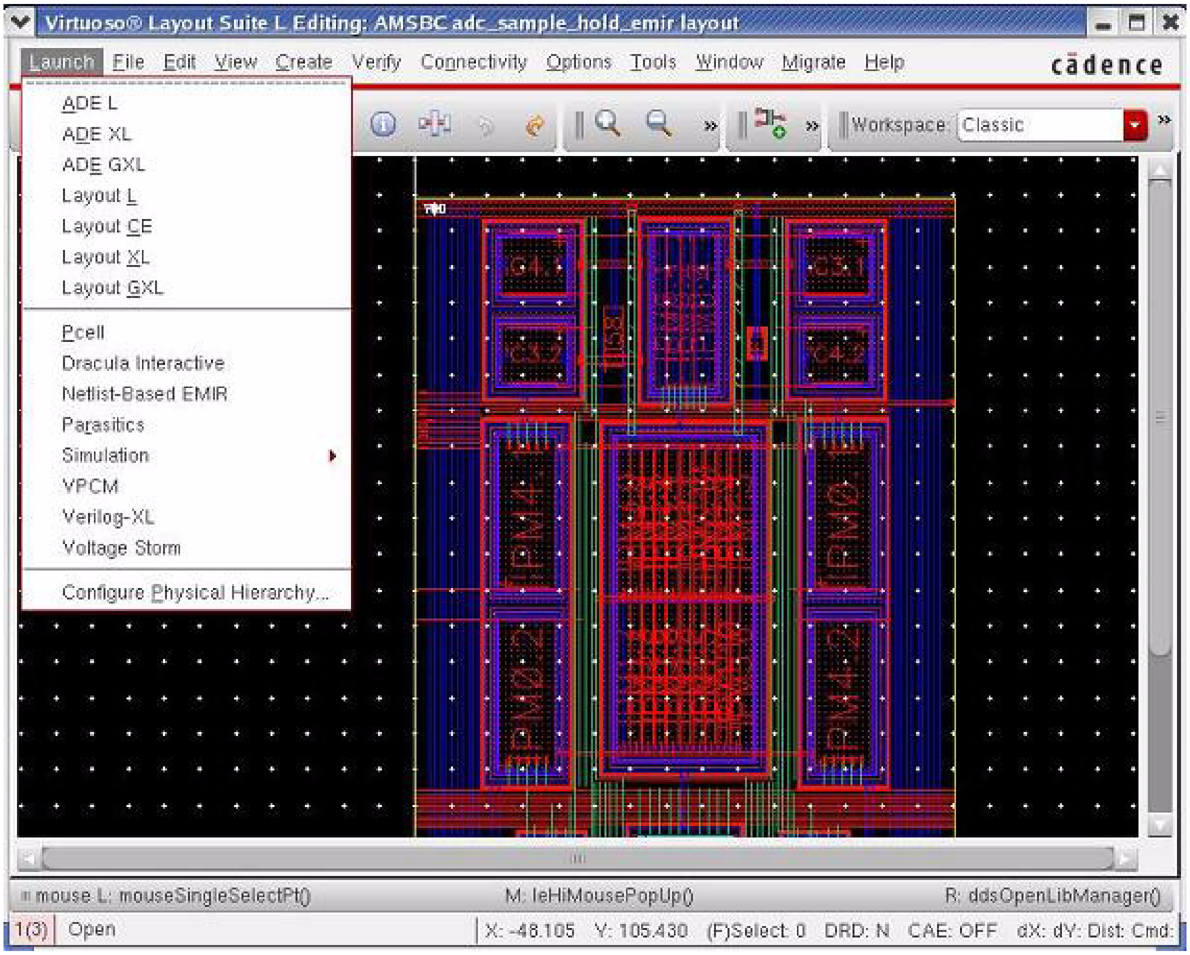
   - `CDSHOME` must be specified in order to display the violation map in the Virtuoso layout editor.

- ❑ In cases where `OA_HOME` must be defined (for example, `setenv OA_HOME /grid/cic/products/dfII/IC6.1.3/lnx86/pink/share/OA`) for Cadence `IC6.X` versions to function correctly in your environment, which is rare, ensure that `OA_HOME` points to the same installation location as specified by `CDSHOME`. The UltraSim netlist based EMIR flow requires `CDSHOME` for GUI display.

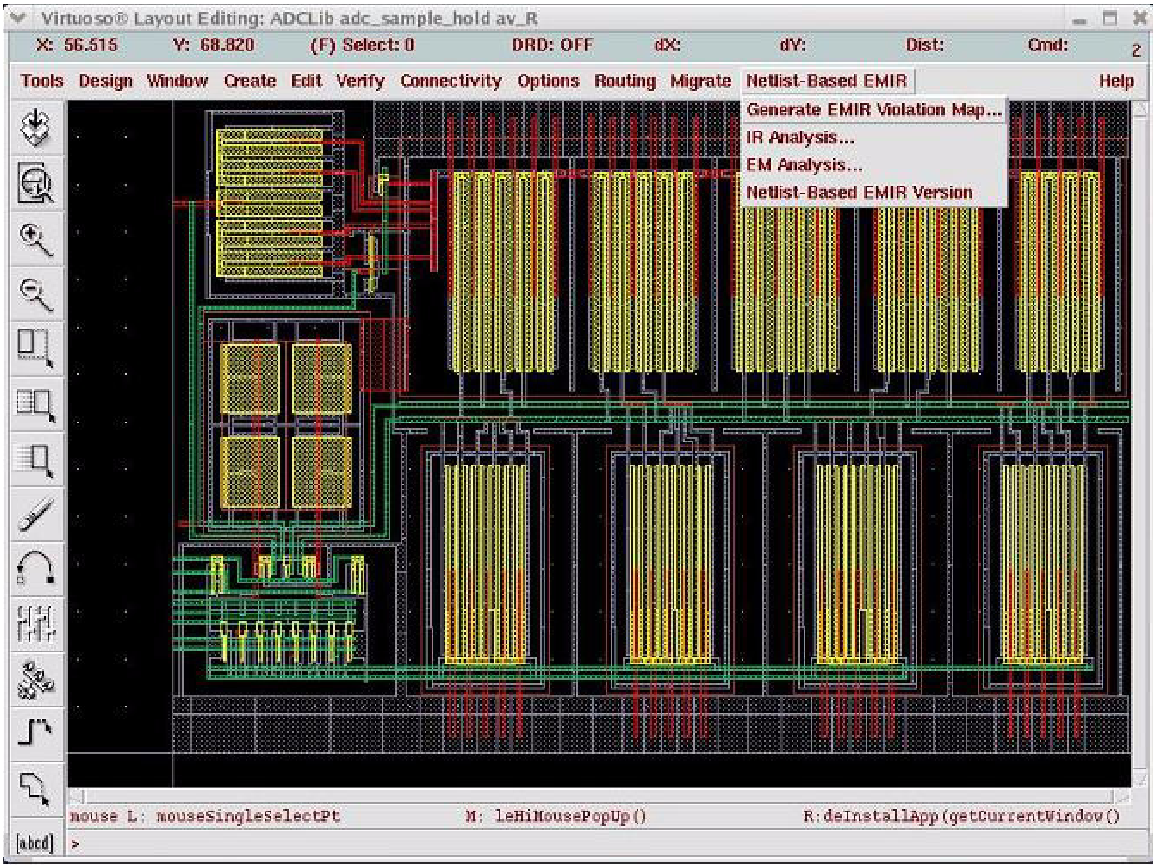
4. Type `virtuoso &` to start the Virtuoso layout editor.

   The command interpreter window (CIW) appears.

5. After opening the desired layout view, choose *Tools – Netlist-Based EMIR* from the main menu bar in the layout window.



6. Select and hold the *Netlist-Based EMIR* menu to display the submenu items.



- ❑ **Generate EMIR Violation Map** converts the intermediate binary data files, saved during the simulation, to CDB or OA database format (see "Violation Maps and Text Reports" on page 531 for more information).

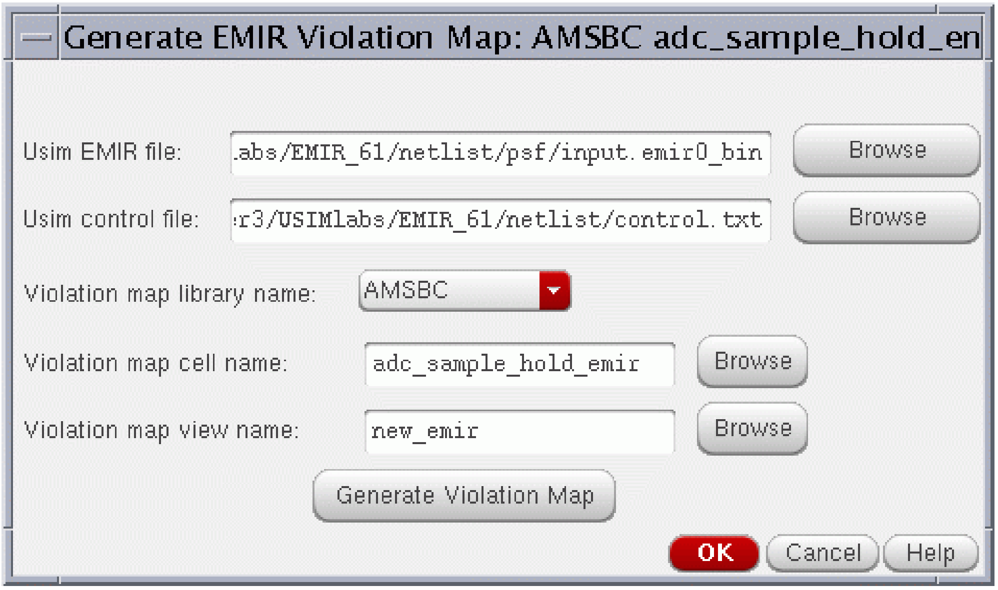
  The commands specified in the control file dictate how the violation maps are generated (see "Control File" on page 543 for more information).

- ❑ **IR Analysis** specifies which violation map to open and allows you to navigate through the IR analysis results. The violation map is a color-coded map of the IR drop analysis results.

- ❑ **EM Analysis** specifies which violation map to open and allows you to navigate through the EM analysis results. The violation map is a color-coded map of the EM analysis results.

- ❑ **Netlist-Based EMIR Version** displays the EM/IR tool version number.

Violation Maps and Text Reports



The Generate EMIR Violation Map form allows you to generate a violation map in DFII format using library, cell, and view names. The *Usim EMIR file* text field corresponds to the full path of the `input.emir0_bin` binary database and the *Usim control file* text field corresponds to the full path of the control file. The *Generate Violation Map* button is used to generate the violation map.

usimEmirUtil Tool

The following command generates violation maps and text reports.

```
usimEmirUtil -layout -format [oa|cdb|none] -db dbFilename -control
control_filename -lib libname -cell cellname -view viewname -text txtfile
-libpath library_path -log emirlog_file
```

The next command generates text reports without violation maps.

```
usimEmirUtil -db dbFilename -control control_filename -text txtfile
```

Note: You must specify `-layout` to generate violation maps, otherwise only text reports are generated, even if `layout format=[cdb|oa]` is specified in the control file.

Description

The `usimEmirUtil` tool is used to post process the binary data file and generate EM/IR violation maps. This tool can be invoked from the command line or in batch mode.

Note: When running the Virtuoso UltraSim netlist-based EM/IR flow in the Cadence Virtuoso environment, `usimEmirUtil` is used.

Arguments

`dbFileName`	Binary data file (`.emir0_bin`)
`control_file name`	Control file name
`libname`	Library name of violation map
`cellname`	Cell name of violation map
`viewname`	View name of violation map
`txtfile`	Text EM/IR reports are renamed using `txtfile` (if `-text txtfile` is not used, the default names `netlist.rpt_ir/netlist.rpt_em` are used instead)
`format`	DFII format of the violation map. One of the following formats can be specified: ■ `oa` ■ `cdb` ■ `none` In GUI mode, the `usimEmirUtil` command automatically detects the DFII format regardless of the "`format=`" statement in the control file. In batch mode, the format specified on the command line will have higher priority than the "`format=`" statement in the control file. When format is not specified on the command line, the software uses the "`format=`" statement in the control file. If the format is neither specified on the command line nor specified in the control file, the default is none, which means that violation maps are not created. This order of priority ensures reuse of the control file between GUI mode and batch mode.
`libpath`	Path of the design library.

`log` EM/IR log file. By default, the software generates the log files based on the format specification as follows:

- For `oa` format, the default name of the EM/IR log file is `usimEmirUtilOA.emirlog`.
- For `cdb` format, the default name of the EM/IR log file is `usimEmirUtilCDB.emirlog`.
- For `none`, the default name of the EM/IR log file is `usimEmirUtilOA.emirlog`.

Note: When there is no CDB or OA database to save, the lib/cell/view arguments are not required.

Example

```
usimEmirUtil -layout -db newemirraw/t3.emir0_bin -control newemirraw/control.txt
-lib myxlib -cell mycell -view emirmap -libpath /usr1/data/AMSADC
```

Sample Text Reports

EM Report

A sample EM text report for net i1.vdd is shown below:

```
----------------------- NET "i1.vdd" -------------------------------
avg

%failed      resistor layer   current  width    pathLength density     limit       needed width/#vias  X1      Y1      X2      Y2
                                (A)    (um)     (um)       (A/um)      (A/um)      (um/#)              (um)    (um)    (um)    (um)
pass-26.42%  rr579    Via2    82.414u  280.000m 155.740    294.336u    400.000u    N/A                 90.040  199.280 90.040  199.280
pass-45.77%  rr583    Via2    728.808u 3.360    155.740    216.907u    400.000u    N/A                 64.050  143.240 64.050  143.240
pass-52.71%  rr581    Via2    635.573u 3.360    155.740    189.159u    400.000u    N/A                 85.810  143.240 85.810  143.240
pass-53.79%  rv394    Cont    43.993u  238.000m 1156.356   184.845u    400.000u    N/A                 90.400  142.720 94.561  56.660
pass-54.71%  rv360    Cont    43.115u  238.000m 315.775    181.154u    400.000u    N/A                 79.560  199.300 79.560  199.300
pass-54.85%  rv388    Cont    42.984u  238.000m 1156.356   180.604u    400.000u    N/A                 90.400  142.720 55.680  56.480
pass-56.78%  rv379    Cont    41.148u  238.000m 1156.356   172.891u    400.000u    N/A                 90.400  142.720 55.680  137.340
pass-60.81%  rv403    Cont    37.309u  238.000m 1156.356   156.762u    400.000u    N/A                 90.400  142.720 94.561  138.300
pass-62.36%  rs334    Via1    84.316u  280.000m 315.775    301.129u    800.000u    N/A                 83.010  199.280 83.010  199.280
pass-63.80%  rg317    Metal3  868.783u 1.200    155.740    723.986u    2.000m      N/A                 64.050  143.240 60.200  156.960
pass-66.09%  rv362    Cont    32.286u  238.000m 315.775    135.655u    400.000u    N/A                 79.560  199.300 66.000  193.920
```

In the above table, each resistor's name, layer, current density, and coordinates (X1, Y1, X2, Y2) are displayed. The `%failed` column shows the violation state. Note that:

- `pathLength` is the total length needed in length based rules.

- `needed width/#vias` shows the necessary width and the necessary vias for a failed metal resistor to pass the EM check. This information is printed for failed resistors only.

IR Report

A sample IR text report is shown below.

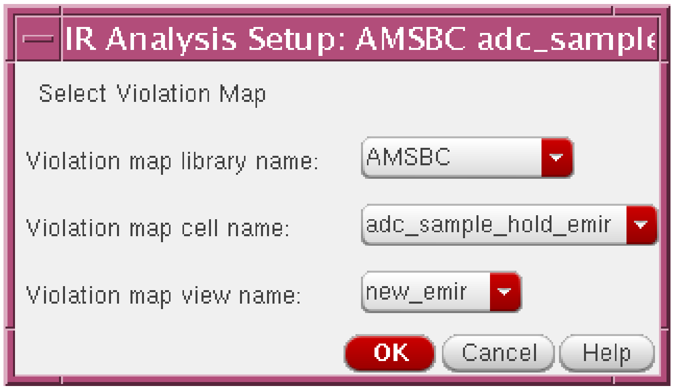
```
*******************************************************
*
*  Copyright(C) 2006, Cadence Design Systems, Inc.
*  Ultrasim EMIR Post Processing Utility, Version 1.1
*
*******************************************************
This file contains the voltage drop results.
RESULTS FILE CREATED = Thu Dec 21 17:38:06 2006
SIMULATOR            = ultrasim
USER SUPPLIED VALUES:
   RESULTS TYPE      = TRANSIENT PEAK
   TRANSIENT START   = 0
   TRANSIENT STOP    = 2e-08
   PIN NAME          = gnd! gnda vdd! vdda
-------------------- "gnda" PIN -----------------------------------
VOLTAGE DROP    NETNAME         TIME            X               Y
  (V)                           (s)            (um)            (um)
50.015n         F423            18.460n        10.160          8.460
42.839n         F417            5.000p         10.630          8.460
```

IR Analysis

To open the IR Analysis Setup form,

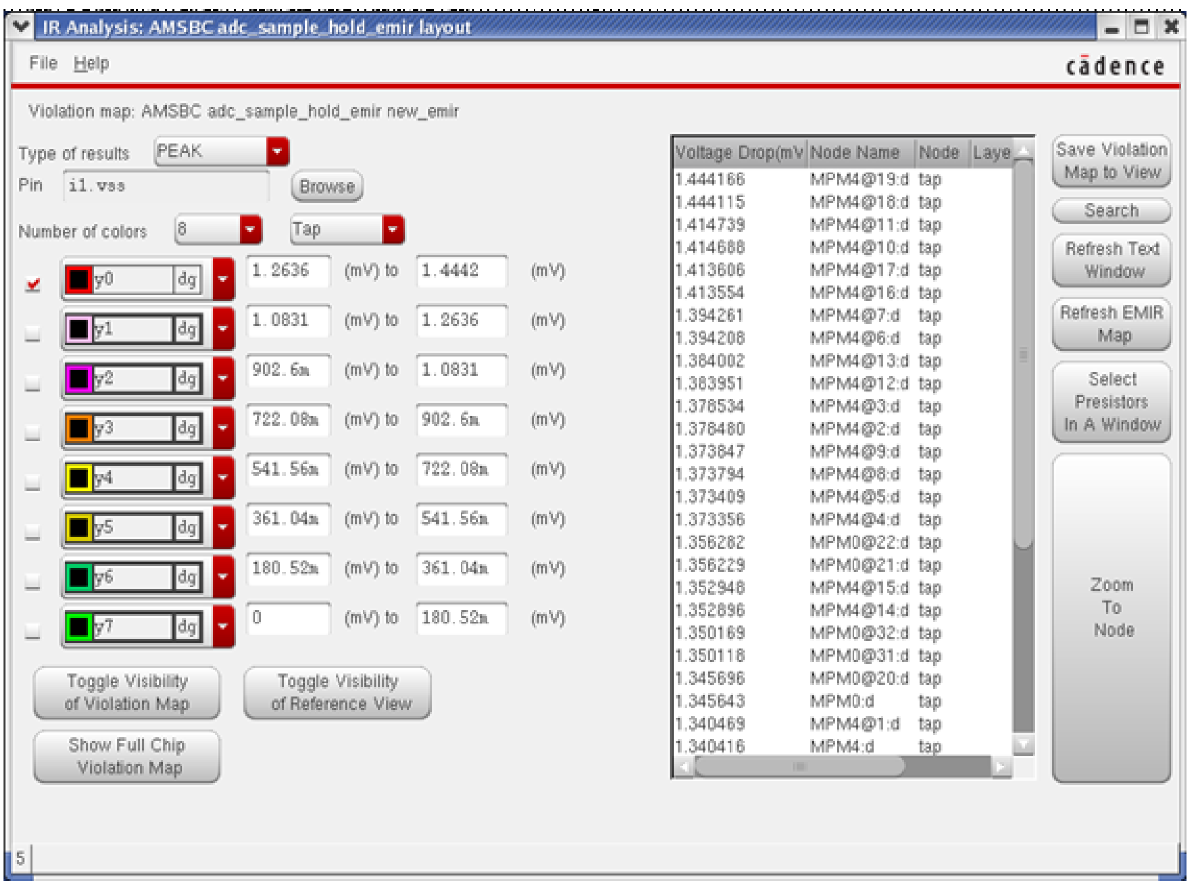
1. Choose *Netlist-Based EMIR – IR Analysis* from the main menu bar in the layout window.

The IR Analysis Setup form appears.



2. Specify a violation map for the IR analysis using the pulldown menus.
3. Click *OK*.

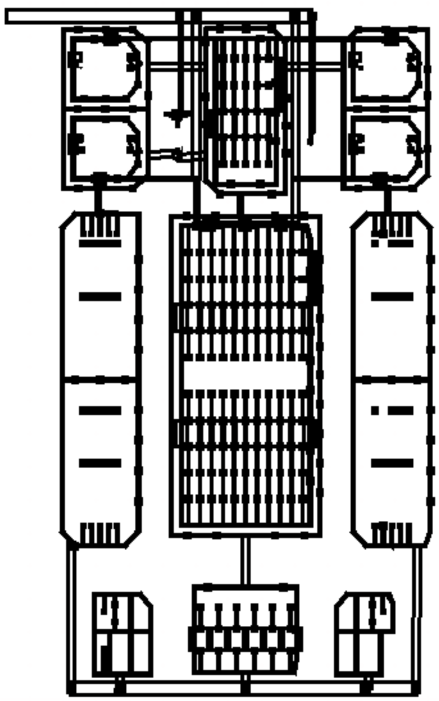
   When the map is successfully loaded, the IR Analysis form appears.



- **Browse** allows you to navigate through different nets displayed in the IR Analysis form (selected net is shown in the form). Multiple nets can be selected. By using the pulldown cyclic button located below the *Browse* button, you can choose one of the following options:
  - *Tap* displays all the tap nodes. These are nodes that are connected to devices. Typically, the tap nodes are denoted by the `*|I` statement in the DSPF file.
  - *Internal* displays all the internal sub-nodes. These are nodes that are not connected to any devices. Typically, the internal sub-nodes are denoted by the `*|S` statement in the DSPF file.
  - *All* displays both tap nodes and sub-nodes.
- **Toggle Visibility of Violation Map** displays the standalone violation map.
- **Toggle Visibility of Reference View** displays the violation map as an overlay on top of the original layout view.
- **Zoom to Node** allows you to cross-probe between the text report and the violation map. To perform this task, select a node in the form text window and click *Zoom to Node*. A magnified view of all resistors connected to the node is displayed in the layout window.

  Note: You might need to zoom out appropriately in order to view all the resistors connected to the node. Resistors (shapes) that are not directly connected to this node are not displayed. To view all the resistors in this region, click the *Toggle Visibility of Violation Map* button.
  - ***Text Report*** is displayed in the text window for the selected pin. The text can be sorted by IR drop value, node name, types of nodes, and time using the pulldown cyclic buttons located at the top of the text window.
- **Save Violation Map to View** allows you to save the violation map for the specified pins in a file.
- **Search** allows you to search in the form text window by node name.
- **Refresh Text Window** refreshes the content in the text window. This button is useful whenever the content of the text window needs to be updated. For example, the color bins are changed; *All* is selected instead of *Tap*, and so on.
- **Refresh EMIR Map** refreshes the content of the violation map displayed in the layout window. This button is useful whenever the content of the displayed violation map needs to be updated. For example, the color bins are changed, different layers are selected, and so on.

- **Select Presistors In A Window** allows you to select a region in the layout window. To select presistors in a layout window, display the violation map in the layout window, click *Select Presistors In A Window*, and select the desired area in the layout window by using the left mouse button. The parasitic resistors located in the text sub-window are highlighted in the report.
- **Show Full Chip Violation Map** allows you to see the violation map for all of the nets in the chip, to help you more readily identify the worst violations.

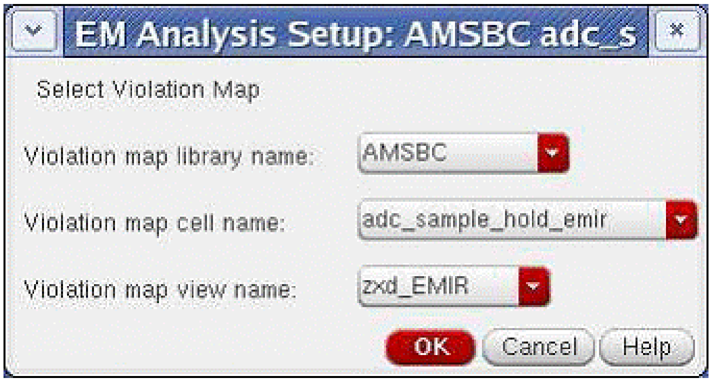


Note: When the layer information for sub-nodes is available in the DSPF/SPEF files, the IR Analysis form displays the layer information just as it appears in the EM Analysis form.

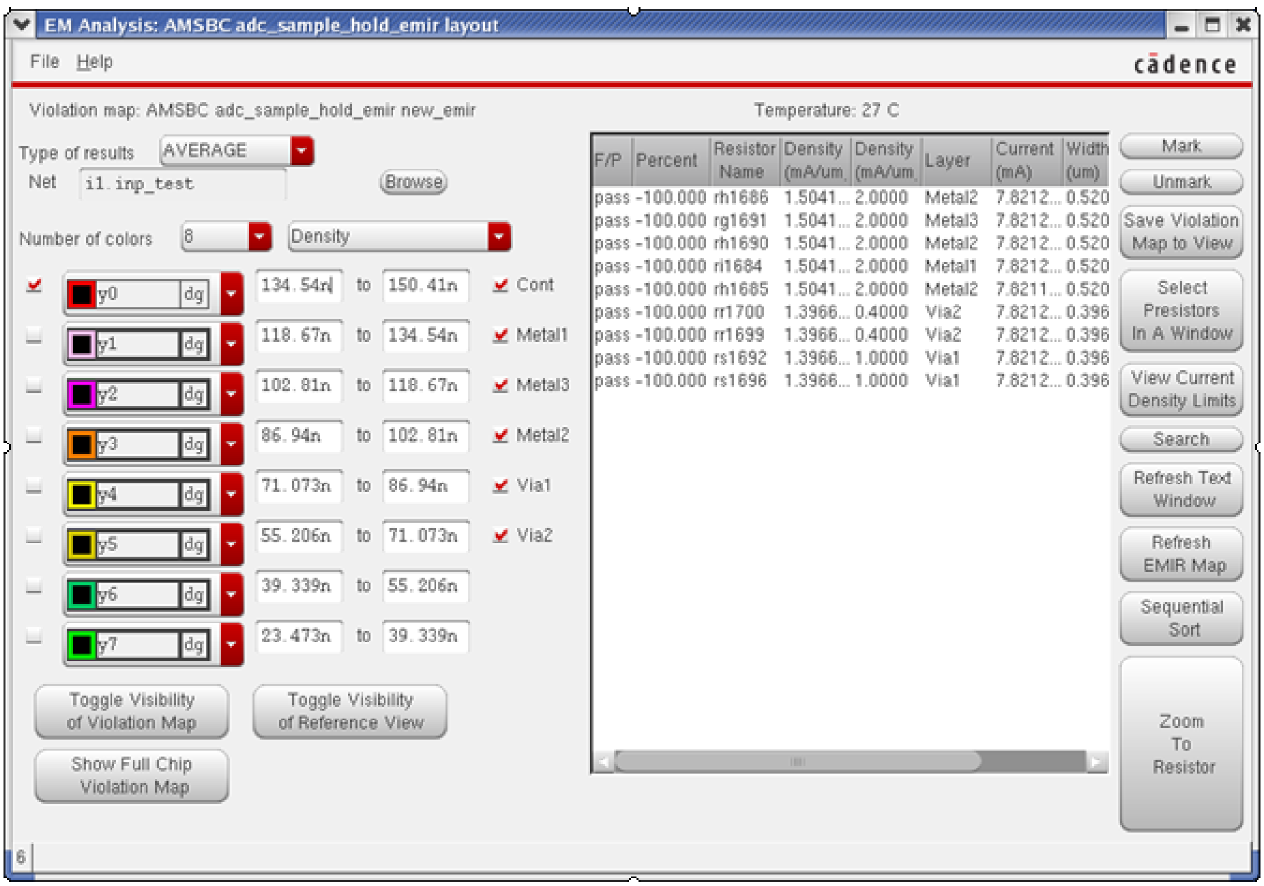
EM Analysis

To open the EM Analysis setup form, choose *Netlist-Based EMIR – EM Analysis* from the main menu in the layout window.

The EM Analysis Setup form appears as shown:



Use this form to specify a violation map for the EM analysis. When the map is successfully loaded in the layout window, the *EM Analysis* form appears. Click *OK* to open the *EM Analysis* form as follows:



■ **Type of results** allows you to view different types of EM analysis results, such as average, peak, and root mean square (RMS) error.

- **Net** allows you to go through the nets.
- **Number of Colors** displays the number of available colors. The number of colors are determined by the `color level=` command in the control file.

Note: You can limit the resistors to be displayed to certain layers by selecting the layer name. To select or deselect a layer, use the checkbox adjacent to the layer name. For example, to display the resistors for the `Via2` layer, select the corresponding checkbox.

- **Temperature** specifies the temperature used for the analysis.
- **Select Presistors In A Window** allows you to select a region in the layout window. To select presistors in a layout window, display the violation map in the layout window, click *Select Presistors in a Window,* and then select the desired area in the layout window by drawing a box in the window. The parasitic resistors located in the text subwindow are highlighted in the report.

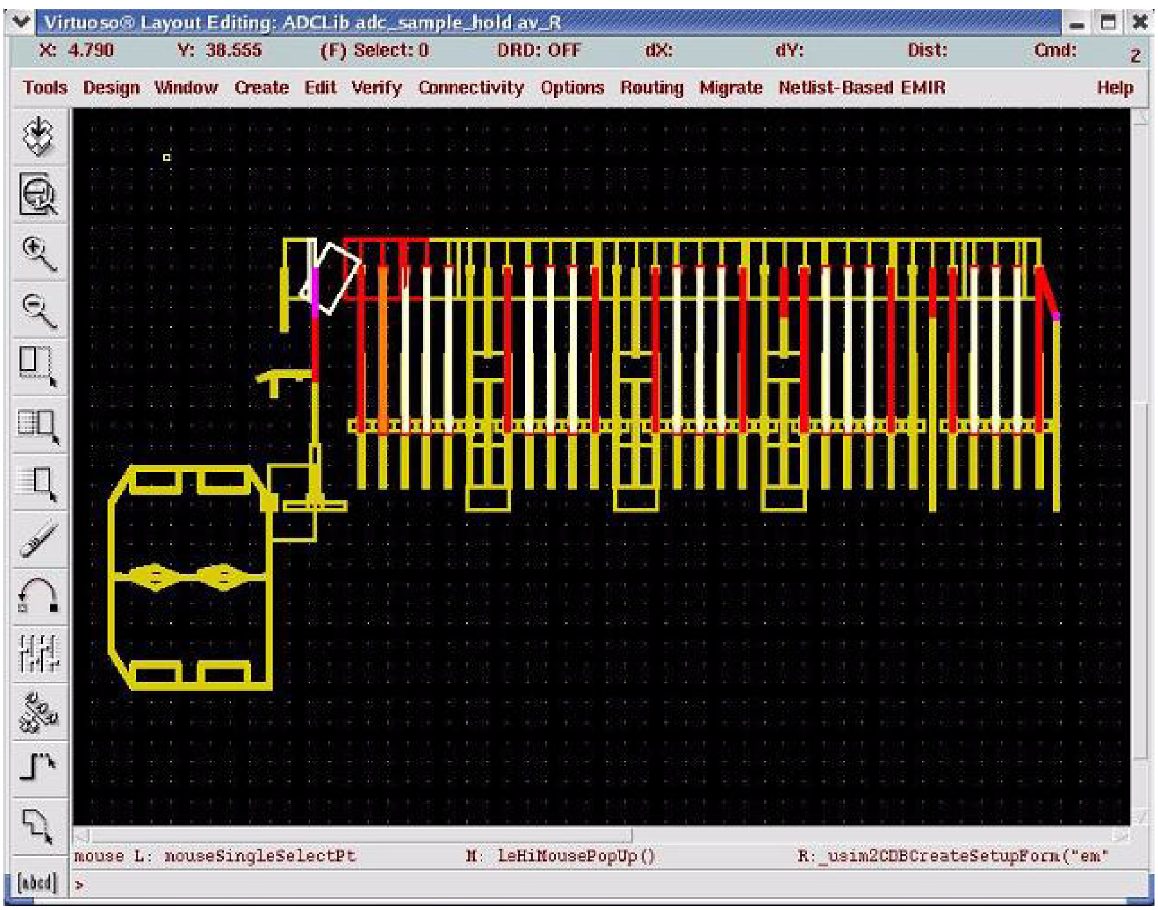
  Note: Before selecting presistors in the layout window, ensure that the violation map is open in the layout window.
- **Mark** places a number (#) sign in front of the selected resistor in the text report.
- **Unmark** removes the number (#) sign from the selected resistor in the text report.
- **View Current Density Limits** allows you to view the EM data file.
- **Show Full Chip Violation Map** displays the full chip EM violation map.
- **Search** allows you to search in the text window by resistor name. The search is case insensitive.
- **Toggle Visibility of Violation Map** displays the standalone violation map.
- **Toggle Visibility of Reference View** displays the violation map as an overlay on top of the original layout view.
- **Zoom to Resistor** allows you to cross probe between the text report and the violation map. To perform this task, select a resistor in the form text window and click *Zoom to Resistor*. A magnified view of all resistors connected to the resistor is displayed in the layout window.
  - ***Text Report*** is displayed in the text window for the selected pin. The text in each column can be sorted by using the pulldown cyclic buttons located at the top of the text window.
- **Refresh Text Window** refreshes the content in the text window. This button is useful whenever the content of the text window needs to be updated. For example, the color bins are changed; *All* is selected instead of *Tap*, and so on.

- **Refresh EMIR Map** refreshes the content of the violation map displayed in the layout window. This button is useful whenever the content of the displayed violation map needs to be updated. For example, the color bins are changed, different layers are selected, and so on.
- **Save Violation Map to View** allows you to save the violation map for the specified pins in a file.
- **Sequential Sort** allows you to perform multi-sorting in the text window. For example, you can first sort the analysis results by resistor name and then sort the results by current.

By default, the current density is used as the criterion for color binning. However, you can choose to specify the pass/fail percentage as the criterion for color binning.

Sample Violation Map

A sample vdd power net violation map is shown below.



Saving Customized Settings in EMIR GUI

You can save your customized settings in both IR Analysis and EM Analysis forms. To save your customized settings in either of these forms, select *File - Save Settings*. The USIM netlist-based EMIR GUI saves `.usimemir.cfg.ir` and `.usimemir.cfg.em` for IR and EM analysis, respectively. These files are saved in the directory using which the layout editor was invoked. The files contain the display settings that were in use at the time the files were saved. When violation maps are loaded again, the saved display settings take effect.

The following information is saved in the `.usimemir.cfg.ir` file for the IR form:

- Number of the color bins.
- The selection of the color bins and their range, respectively.
- The selection of layer filtering, if applicable.
- The selection of the node types that are displayed. The node types could be "Tap", Internal Subnodes, or "All" (includes both tap nodes and internal subnodes).
- The selection of the analysis type, which could be AVERAGE, PEAK, RMS or CUSTOM.

The following information is saved in the `.usimemir.cfg.em` file for the EM form:

- Number of colors.
- The selection of the color bin and their range, respectively.
- The selection of the layer filtering.
- The selection of the quantity used in color binning, that is, the pass or fail percentage of current density.
- The selection of analysis type, which could be AVERAGE, PEAK, RMS or CUSTOM.

Note: It is recommended that you first find the desired display settings and save them by selecting File - Save Settings. Next, display the violation map again for the customized settings to take effect.

A sample of the `.usimemir.cfg.ir` file settings is as follows:

```
_usim2CDBGlobalLoadIrSetting=make_usim2CDBLoadIrSetting()
_usim2CDBGlobalLoadIrSetting->layerRange = makeTable("_usim2CDBGlobalLayerRange")
_usim2CDBGlobalLoadIrSetting->layerRange["0,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["4,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["4,layer"]="y4"
_usim2CDBGlobalLoadIrSetting->layerRange["7,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["7,layer"]="y7"
```

```
_usim2CDBGlobalLoadIrSetting->layerRange["6,layer"]="y6"
_usim2CDBGlobalLoadIrSetting->layerRange["1,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["6,range"]=list(0.048173 0.096347)
_usim2CDBGlobalLoadIrSetting->layerRange["7,range"]=list(0.0 0.048173)
_usim2CDBGlobalLoadIrSetting->layerRange["layers"]=list("y0" "y1" "y2" "y3" "y4"
"y5" "y6" "y7")
_usim2CDBGlobalLoadIrSetting->layerRange["0,range"]=list(0.33721 0.3853875)
_usim2CDBGlobalLoadIrSetting->layerRange["1,layer"]="y1"
_usim2CDBGlobalLoadIrSetting->layerRange["5,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["0,layer"]="y0"
_usim2CDBGlobalLoadIrSetting->layerRange["3,range"]=list(0.19269 0.24087)
_usim2CDBGlobalLoadIrSetting->layerRange["3,layer"]="y3"
_usim2CDBGlobalLoadIrSetting->layerRange["5,range"]=list(0.096347 0.14452)
_usim2CDBGlobalLoadIrSetting->layerRange["5,layer"]="y5"
_usim2CDBGlobalLoadIrSetting->layerRange["4,range"]=list(0.14452 0.19269)
_usim2CDBGlobalLoadIrSetting->layerRange["2,layer"]="y2"
_usim2CDBGlobalLoadIrSetting->layerRange["1,range"]=list(0.28904 0.33721)
_usim2CDBGlobalLoadIrSetting->layerRange["2,range"]=list(0.24087 0.28904)
_usim2CDBGlobalLoadIrSetting->layerRange["3,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["2,isValid"]=t
_usim2CDBGlobalLoadIrSetting->layerRange["6,isValid"]=t
_usim2CDBGlobalLoadIrSetting->analysis = "PEAK"
_usim2CDBGlobalLoadIrSetting->numColorLevel = 8
_usim2CDBGlobalLoadIrSetting->nodeType = "Tap
```

Note: Modifying the `.usimemir.cfg.em` and `.usimemir.cfg.ir` files is not recommended.

Command Syntax

Control File

Note: The plus (+) symbol is used for line continuation and the semicolon (;) symbol is used for comments.

Table 9-1 Control File Command Syntax

Command	Description

`color level`	Syntax:
	`color level=value`
	The `color level` command is used to control the number of colors displayed in the violation map. The usimEmirUtil Tool supports a maximum of 10 color levels. The tool calculates the levels automatically, according to the limits specified in `emdata file` (for example, `avgCurrentDensSpecList` and `rmsCurrentDensSpecList`), and generates the appropriate violation map. The default value is 10 color levels.
	For example,
	`color level=9`
	tells the Virtuoso UltraSim simulator to display the violation map with up to nine levels of colors.
	Note: Make sure that you specify the control file using the `file` argument in the `.usim_emir` command.

Table 9-1 Control File Command Syntax, *continued*

Command	Description

`layout format`	Syntax:
	`layout format=[cdb|oa|none]`
	The `layout format` command is used to specify which type of format is used for the violation map (default is `none` which specifies that violation maps are not generated). The Virtuoso UltraSim simulator supports CDB or OA database formats.
	To generate a violation map, you need to specify `usim_emir format=[layout]` in the netlist file or `usim_emir type=power` or `type=signal`. This allows the simulator to save the geometry information, and voltages/currents of nodes and resistors, into databases.
	For example, in the netlist file, the following statement is specified as:
	`usim_emir type=all format=[layout]`
	In the control file, it is specified as:
	`layout format=[cdb]`
	Note: The `layout format` command also interacts with the `-format` option of `usimEmirUtil`.

`emdata file`	Syntax: `emdata file='filename'` The `emdata file` command specifies the path of the EM data file, which contains the current density limits information for each layer. The file syntax is the same as the Virtuoso Analog VoltageStorm ElectronStorm Option (VAEO) product syntax. **Note:** The `emdata file` command syntax is case-sensitive.

Table 9-1 Control File Command Syntax, *continued*

Command	Description

`pwnet net`	Syntax:
	`pwnet net=net1 analysis=[vmax vavg vrms imax iavg irms icustom] vref=value1 net=net2 analysis=[vmax vavg vrms imax iavg irms icustom] vref=value2`
	The `pwnet net` command tells the `usimEmirUtil` tool that specified nets are power nets (no default for `pwnet net`). The following types of analysis can be performed: `vmax`, `vavg`, `vrms`, `imax`, `iavg`, `irms`, and `icustom`. The automatically detected reference voltage is used by the `usimEmirUtil` tool for `vmax`, `vavg`, and `vrms` analysis. The reference voltage can be specified by using `vref` as well. `vref` is only recommended when the net is not connected to ideal voltage source.
	For example,
	`pwnet net=VDD analysis=[vmax] net=VSS analysis=[vmax irms iavg]`
	tells the `usimEmirUtil` tool that for the VDD power net, peak IR drop analysis is required, and for the VSS power net, peak IR drop, average, and RMS EM analyses are required.

`signal net`	Syntax:
	`signal net=net1 analysis=[vmax vavg vrms imax irms iavg icustom] vref=value1 net=net2 analysis=[vmax vavg vrms imax irms iavg icustom] vref=value2`
	The `signal net` command tells the `usimEmirUtil` tool that specified nets are signal nets (no default for `signal net`). The following analysis can be performed: `vmax`, `vavg`, `vrms`, `imax`, `iavg`, `irms`, and `icustom`. If nets are not specified, then all of the auto-detected power nets are reported.
	You need to specify the reference voltage for `usimEmirUtil` to calculate the `vmax`, `vavg` and `vrms`. The reference voltage can be specified by using `vref` as well. `vref` is only recommended when the net is not connected to ideal voltage source.
	For example:
	`signal net=netA analysis=[vmax] vref=3`
	tells the UltraSim software to calculate peak voltage drop for `netA` with reference voltage of `3V`.

Table 9-1 Control File Command Syntax, *continued*

Command	Description
`report text`	Syntax: `report text=0|1` The `report text` command is used to enable the `usimEmirUtil` tool to compute text reports from the binary database (`report text=1`). The `design_name.rpt_ir` and `design_name.rpt_em` text reports are generated. If `report text=0` is selected, text reports are not generated from the binary database. **Note:** Use the `usimEmUtil` command line tool if you want to specify the name for the EM and IR reports (see "usimEmirUtil Tool" on page 531 for more information).

`warnmsg limit`	Syntax:
	`warnmsg limit=value Warning_Type1 Warning_Type2`
	The `warnmsg limit` command is used to limit the number of warning messages by classifying warning messages in the `usimEmirUtil` tool to a different catalog (for example, EMIR-1 EMIR-2).
	For example,
	`warnmsg_limit=50 EMIR-1 EMIR-2`
	tells the simulator to limit the number of `EMIR-1` and `EMIR-2` type warning messages to a maximum of 50 messages.

Table 9-1 Control File Command Syntax, *continued*

Command	Description

`geounit filename`	Syntax:
	`geounit filename="filename" unit=value`
	The `geounit filename` command allows you to designate the units used for the geometry information for the net section in the specified parasitic files (default for units is1 um). This does not apply to device parameters. In addition to using `geounit` in the control file, it is necessary to add the `file=<control-file>` statement in the `usim_emir` commands during simulation for `geounit` to work properly
	For example,
	`geounit filename="file1.dspf" unit=1m`
	tells the simulator to use the 1 mini-meter geometry unit for `file1.dspf`.
	Note: For `geounit` to work as expected, ensure that you add `file=control_file` option in the `usim_emir` statement during simulation.

Table 9-1 Control File Command Syntax, *continued*

Command	Description
`use via_res`	Syntax: `use via_res=0|1` Default is `1`. Often, each via or contact resistor consists of a number of minimum vias or contacts, which are called via arrays or contact arrays. The width and length of the minimum via and contact are defined in the EM data file by commands `viaWidthList` and `viaLengthList`, respectively. If the length of minimum via/contact is not defined, the minimum length is assumed to be equal to the minimum width. UltraSim must calculate the number of vias/contacts in an array to check EM violation and draw this resistor in the violation map. UltraSim supports two ways to calculate the number of vias: ■ When `via_res=1`: The number of vias is calculated by resistance relationship, that is, it is equal to the ratio of minimum resistance and the resistor's value. ■ When `via_res=0`: UltraSim uses the via/contact area or the width specified in the DSPF file. The number of vias is the ratio of the resistor area and the minimum via area. **Notes**: If the via layer's EM rule is width-dependent, UltraSim uses the total width to check EM violation. The total width is the sum of the width of all the minimum vias in the array, assuming all the vias in the via array are aligned in one line. In EM violation maps, it is assumed that all the minimum vias in the via array are arranged in square. For example, if the calculated number of vias is four, they are assumed to be arranged in 2x2 fashion. If the minimum via width is set to be 1um and minimum via length is set to be 2um, then this via resistor is drawn as rectangle with width of 2u and length of 4u. If the shape of the via resistor can't be determined properly, for instance, `viaWidthList` is not defined, the via is not drawn.

Table 9-1 Control File Command Syntax, *continued*

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td></td><td>The shape of the via resistor in IR violation maps is consistent with that in EM violation maps. If EM violation maps are impossible to determine, for example, no EM rule files is specified, (this could happen when you are only interested in IR analysis), in which case, the via shapes are assumed to be square, its width would be equal to the specified width in the DSPF file or to the calculated width= sqrt(area), where area is the specified area for this via resistor in the DSPF file. If the shape of the via can't determined properly, the via is not drawn in the violation maps.

Example,

<code>use via_res=0</code>

tells the simulator to use the area specified in the DSPF file to calculate the number of vias.</td></tr>
<tr><td><code>userDefinedCurrentCalc</code>

<code>file=”file_name”</code>

<code>proc=”procedure_name”</code></td><td>Syntax:

<code>userDefinedCurrentCalc file= “file_name”</code>
<code>proc="procedure_name"</code>

This command is used to support customized current calculation. The customized current calculation method is specified in SKILL code.

In this command:

■ <code>userDefinedCurrentCalc</code> is the keyword.
■ <code>file</code> is the file name that contains the skill code.
■ <code>proc</code> is the SKILL procedure used for current calculation.

Note: To enable custom current analysis, the analysis type must be specified as <code>icustom</code>. The corresponding EM rule is specified in a section called “<code>userDefinedCurrentDensSpecList</code>” in the EM data file. For more information, see <u>Customized Current Calculation and EM Check</u> on page 559.

For example,

<code>userDefinedcurrentCalc file= “myskill.il” proc= “Icalc”</code>

tells the simulator that the <code>myskill.il</code> files contains the skill code and <code>Icalc</code> is the procedure that will be used for current calculation.</td></tr>
</table>

Table 9-1 Control File Command Syntax, *continued*

Command	Description

`ir_percent`
 	
`ir_threshold`
	Syntax:
	
	`ir_percent=value`
	
	`ir_threshold=value`
	
	These commands are used to limit the resistors and subnodes saved in violation maps and text reports generated by `usimEmirUtil`.
	
	For example,
	
	`ir_percent=5`
	
	tells `usimEmirUtil` to save the subnodes whose IR drops are among the top 5 percent IR drops in the violation maps and text reports.
	
	`ir_threshold=0.01`
	
	tells `usimEmirUtil` to save the subnodes whose IR drops are over 0.01v drop in the violation maps and text reports.

Table 9-1 Control File Command Syntax, *continued*

Command	Description

`splitvmap`	Syntax:
	
	`splitvmap net=netname1 byanalysis=yes | no bylayers=yes | or`
	
	`+ net=netname2 byanalysis=yes | no bylayers=yes | no`
	
	This command instructs `usimEmirUtil` to generate separate violation maps for the specified nets. The number of separate violation maps can be further determined by the arguments, `bylayers` and `byanalysis`. When `byanalysis` is set to `yes`, the violation maps for the specified nets will be further separated by the analysis type. When `bylayers` is set to `yes`, the violation maps for the specified nets will be further separated by the layers. The default setting for `byanalysis` and `bylayers` is `no`. There is no default value for net and wildcards are supported for net names. For the nets that are not specified, `usimEmirUtil` generates one violation map. When the `splitvmap` command is not specified, `usimEmirUtil` generates one violation map for all the nets. The naming convention for the multiple violation maps is as follows:
	
	`rootname_netname_analysis_layer`
	
	The netname, analysis and layer name are appended to the root name that you specify using the `-view` parameter of `usimEmirUtil`.
	
	Consider that the root view name is EMIR, you have splitvmap net=VDD byanalysis=yes bylayers=yes net=VSS byanalysis=yes in the control file, there are only metal1 and metal2 layers in VDD, VDD has Iavg and Imax analysis, VSS has Iavg and Imax analysis, and there are other nets to be analyzed. Then, the following violation maps are generated:
	
	`EMIR --- this includes the violation map for all the nets except VDD and VSS.`
	
	`EMIR_VDD_Imax_metal1 - this includes the violation map for layer metal1 of VDD's Imax`
	
	`EMIR_VDD_Iavg_metal1`
	
	`EMIR_VDD_Imax_metal2`
	
	`EMIR_VDD_Iavg_metal2`
	
	`EMIR_VSS_Imax - this will include violation map of VSS's Imax`
	
	`EMIR_VSS_Iavg - this will include violation map of VSS's Iavg.`

Table 9-1 Control File Command Syntax, *continued*

Command	Description

`splitreport`	Syntax:
	`net=netname1 byanalysis=yes | no bylayers=yes | or`
	`+ net=netname2 byanalysis=yes | no  bylayers=yes | no`
	The command instructs `usimEmirUtil` to generate separate textual reports for the specified nets. The number of separate textual reports can be further determined by the arguments, `bylayers` and `byanalysis`. When `byanalysis` is set to `yes`, the textual reports for the specified nets will be further separated by the analysis type. When `bylayer` is set to `yes`, the textual reports for the specified nets will be further separated by the layers. The default value for `byanalysis` and `bylayers` is `no`. There is no default for net and wildcards are supported for net names. For the nets that are not specified, `usimEmirUtil` generates one textual report. When the `splitreport` command is not specified, `usimEmirUtil` generates one textual report for all the nets. The naming convention for the textual report is as follows:
	`rootname_netname_analysis_layer`
	The netnames, analysis, and layer names are appended to the root name. The root name can be specified by using the `-txtfile` parameter of `usimEmirUtil`.
	Consider that the root textual report name is "report", the control file has the `splitreport net=VDD  byanalysis=yes bylayers=yes net=VSS byanalysis=yes` statement, there are other nets for EM analysis only, there are only metal1 and metal2 layers on VDD, VDD has Imax and Vmax analysis, and VSS has Vmax analysis only. The following report is generated:
	`report.rpt_em`
	`report_vdd_vmax_metal1.rpt_ir`
	`report_vdd_vmax_metal2.rpr_ir`
	`report_vdd_imax_metal1.rpt_em`
	`report_vdd_imax_metal2.rpt_em`
	`report_vss_vmax.rpt_ir`

Here is a sample control file:

```
color level=8
layout format=[cdb]
```

```
pwnet net=[i1.vdd] analysis=[vmax iavg]
+ net=[i1.vss] analysis=[vmax iavg]
signal net=[i1.*] analysis=[iavg]
emdata file="emDataFile.txt"
report text=1
```

EM Data File

An EM data input file needs to be created to specify technology information, such as current density limits, and to map between the layers for highlighting. The file syntax is the same as the VAEO product syntax (refer to the *VoltageStorm Transistor-Level PGS User Guide* for more information).

Note: The EM data file syntax is case-sensitive. The file must start with parenthesis.

Table 9-2 EM Data File Command Syntax

Command	Description
`semicolon (;)`	Single line comments can be added by starting the line with a semicolon (`;`). For example, `; This is a comment line`
`currentDensityMPV`	The `currentDensityMPV` (current density milliamps per via) command allows you to define the current density specifications in terms of milliamps per via instead of the standard milliamps per micron (most foundries specify via and contact current densities in milliamps per via). For example, `currentDensityMPV=true` **Note:** If `currentDensityMPV` is used, then all current density specifications for vias must be stated in terms of milliamps per via.

Table 9-2 EM Data File Command Syntax, *continued*

Command	Description
`junctionTemp`	The `junctionTemp` (junction temperature) command can be specified if you want to use `deltaT` to define the value of current density as a function of the delta between the simulation temperature and the specified junction temperature. A better way is to define `deltaT` directly. For example, `junctionTemp=110` **Note:** When the simulation temperature is not within the temperature values specified in the EM file, the simulator chooses the EM limit that corresponds to the nearest temperature. When the simulation temperature is within the temperature value specified in the EM rule file, the simulator performs linear interpretation to get the EM limit.
`deltaT`	The `deltaT` command is directly supported in expression for EM limits. `deltaT` is often used in some foundry's rms EM rule. `deltaT` must be positive. **Note:** Do not use `junctionTemp` and `deltaT` together. `junctionTemp` will be phased out gradually.
`minI`	The `minI` (minimum current) command can be used to limit reporting of small currents which are of little or no interest during either power or signal EM analysis. The value of `minI` is specified in amps (A). Any wire segments with current values less than `minI` are not displayed in the EM results windows or reported in the EM results file. The default value of `minI` is 0 and the analysis is not affected by this command. For example, `minI=1e-9`

Table 9-2 EM Data File Command Syntax, *continued*

Command	Description
`relminJ`	The `relminJ` (relative minimum current density) command can be used to limit reporting of small current density compared to the density limits. Any wire segments with a current density value less then `relminJ` of its density limit specified in `emdata file` are not reported in the EM text report or violation map. The default value of `relminJ` is 0 and the analysis is not affected by this command. For example, `relminJ=0.01`
`routingLayers`	The `routingLayers` (routing layers) command is used to specify the names of the routing layers. `currentDensityName` is used. For information about `currentDensityName`, see Cross Reference Layers. **Note:** Includes only poly and metal layers. For example, `routingLayers = ("poly1" "m1" "m2" "m3" "m4" "m5" "m6" "m7" "m8" "m9" "MD")`

`viaLayers`	The `viaLayers` (via layers) command is used to specify the names of via layers. `currentDensityName` is used. For information about `currentDensityName`, see Cross Reference Layers.
	
	For example,
	
	`(Metal1 Cont Poly) // the via layer is Cont.`
	
	In the next example,
	
	`viaLayers = ( "cw" "v1" "v2" "v3" "v4" "v5" "v6" "v7" "v8")`
	
	shows the `viaLayers` syntax.

Table 9-2 EM Data File Command Syntax, *continued*

Command	Description
`viaWidthList`	The `viaWidthList` (via width list) command specifies the minimum via width. For example, `viaWidthList = ( ( "Cont" 0.2 ) ( "Via1" 0.2 ) )` The name of the via layer is `Cont` in `currentDensityName` space and the minimum width of the default via for this layer is `0.2`. This information is normally specified in the Cadence DFII technology file and must also be included in the EM data file. **Note:** The DFII technology file must include all of the vias specified in the `viaLayers` section of the EM data file.
`viaLengthList`	The `viaLengthList` (via length list) command specifies the via length for use in calculating the area of vias that are not square. This feature is only required when `currentDensityMPV` is set to `true`. If the vias in the chip design are square-shaped, then you can specify `viaWidthList` without `viaLengthList`. For example, `viaLengthList = (Via1 0.12)`
`CurrentDensSpecList`	The `CurrentDensSpecList` (current density specifications) command refers to the current density specification declared by the foundry. This syntax allows one layer to support different current density specifications according to the width of the material. For the Virtuoso UltraSim netlist-based EM/IR flow, the current density specifications in `emdata file` include `avgCurrentDensSpecList`, `rmsCurrentDensSpecList`, and `peakCurrentDensSpecList`, and `userDefinedCurrentDensSpecList`. **Note:** Each specification in the EM rule file can be unique with its own current density and temperature. See "Current Density Specification Example" on page 557 for an example of current density specifications.

Current Density Specification Example

The following is an example of a current density specification (`CurrentDensSpecList`) for a given layer.

```
(nil layer "poly1" minW 0.0 maxW -1 res 0.68 currentDensity ((1.2100,100)
(1.2110,110) (1.2125,125)))
```

- **nil** is reserved and indicates the start of the line.
- **layer** is followed by the specific layer name used to apply the current density line. The layer name is the current density layer name from the layer cross-reference section of the EM data file.

  For example,

  ```
  layer “poly1”
  ```

- **minW** specifies the minimum line width in microns (`minW` <= width).

  If `minW` is specified for vias, the calculated width is the total width of the vias assuming they are laid in one long row.

- **maxW** specifies the maximum line width in microns (width < `maxW`). When set to -1.0, `maxW` represents infinity.

  If `minW` and `maxW` values are specified for vias, the calculated width is the total width of the vias assuming they are laid in one long row.

- **minL** specifies the minimum line length in microns (`minW` <= length). `minL` is optional.
- **maxL** specifies the maximum line length in microns (`minW` <= length). `maxL` is optional.
- **res** specifies the resistance in ohms. For contacts and via layers, `res` should be set to the total resistance of a minimum width contact.
- **via_range** specifies the number of vias in a via array. In some technologies, for vias, the current density limit may vary depending on the number of vias in a via array. For example, the current density limit for *via1* at 110 degree can be as below:

  If number of vias in an array is 1, current density limit = 0.8 mA/um
  If number of vias in an array is 2 or 3, current density limit = 1.0 mA/um
  If number of vias in an array is 4 or 5, current density limit = 2.0 mA/um
  If number of vias in an array is over 5, current density limit = 0.4 mA/um

  then, the EM data file that represents the above is:

  ```
  (nil layer "via1" minW 0.0 maxW -1.0 res 1.4 currentDensity ((0.4 , 110)))
  (nil layer "via1" minW 0.0 maxW -1.0 res 1.4 via_range 1 currentDensity ((0.8 , 110)))
  (nil layer "via1" minW 0.0 maxW -1.0 res 1.4 via_range 3 currentDensity ((1.0 , 110)))
  (nil layer "via1" minW 0.0 maxW -1.0 res 1.4 via_range 5 currentDensity ((2.0 , 110)))
  ```

via_range is optional in the EM data file.

- **currentDensity** is specified in milliamps per micron at a specified simulation temperature (`temperature` is specified in degrees Celsius).

  For example,

  ```
  currentDensity ((1.2100,100) (1.2110,110) (1.2125,125))
  ```

  Most foundries specify via and contact current densities in milliamps per contact or square micron. The foundry specification must be converted to milliamps per micron. For example, if the foundry specification provides a current density of 0.4 mA/via and the via is 0.2 microns multiplied by 0.2 microns, then the `currentDensity` value is 0.4/0.2=2 mA/um.

Tip

Use the `currentDensityMPV` command to specify the current densities in milliamps per contact as provided by the foundry, so manual conversion is not required.

You can specify `currentDensity` as an equation containing the variables `w` for width, `l` for length, and `deltaT` for temperature difference.

For example,

```
currentDensity ( (14*w*deltaT,100) (1.434 *3.9*w, 110) (5.3,125))
currentDensity ( (14*w*deltaT/l,100) (1.434 *3.9*w*l, 110) (5.3,125))
```

You do not need to specify the value of `w` because it is located in the actual design data. Current density equations using `w` are evaluated for each metal line width. For example, a `m1` layer has a different current density specification for line widths 0.16 um versus 0.25 um. When an equation is used in the `currentDensityspec`, `unknown` may be written to the results of the analysis if the equation cannot be resolved.

To use `deltaT` to define current density for `irms` analysis, you can define `deltaT` directly. For example:

```
deltaT=5
```

You can also specify `currentDensity` as an equation using the following operators: `+`, `-`, `*`, `/`, `sqrt()`, and `exp()`.

For example,

```
currentDensity ( (sqrt(4.9100),100) (4.9110,110) (4.9125,125))
```

The square root function takes a positive floating point number as input.

If multiple (`currentDensity`, `temperature`) pairs are specified in `CurrentDensSpecList`, then you must ensure that current densities are defined at the specified temperatures for all the layers in one construct using `avgCurrentDensSpecList`, `rmsCurrentDensSpecList`, or `peakCurrentDensSpecList`.

Important

Failing to define the current densities at specified temperatures for all layers in one construct results in an error that stops the analysis.

The following is an example that uses the `peakCurrentDensSpecList` parameter.

```
peakCurrentDensSpecList = (
(nil layer "m1" minW 0.0 maxW 0.16 currentDensity ( (1.434*3.9*w,100) (3.9,110)
(0.387*3.9,125)))
(nil layer "m1" minW 0.16 maxW -1.0 currentDensity ( (14*w*deltaT,100)
(7.0-0.16,110) (5.3,125)))
```

The `currentDensity` for layer `m1` is calculated using the actual line width, and is dependent on whether it is greater than the 0.16 `maxW` specified as the break point between the two lines.

UltraSim also supports simulation temperature `T` in the expression of the EM rules. For example, if the average current density limit for Metal1 is `Idc = exp(9495/T-25.45)*2.34e-3*(w-0.042)` and `2.0*T` for Poly, the corresponding EM rule is:

```
avgCurrentDensSpecList = (
(nil layer "Poly"   minW 0.0 maxW -1.0 currentDensity ((xp(9495/T-25.45)*2.34e-
3*(w-0.042) , T)))
(nil layer "Metal1" minW 0.0 maxW -1.0 currentDensity ((2.0*T , T)))
```

where, `T` is the simulation temperature. UltraSim gets value of `T` from the simulation database.

Customized Current Calculation and EM Check

To enable customized current calculation and EM check, you need to specify the `icustom` analysis type in the control file. This signifies that the current calculation is customized. When `icustom` is specified, `usimEmirUtil` searches for the procedure defined using the `userDefinedCurrentCalc` command and calculates the current followed by the current density. If `icustom` is not specified, `userDefinedCurrentCalc` does not take effect.

You can specify the location of the skill procedure in the control file as follows:

```
userDefinedCurrentCalc file=”file_name” proc="procedure_name"
```

The calculated current density is checked against the EM rule defined by the `userDefinedCurrentDensSpecList` section in the EM data file.

Cross Reference Layers

The cross reference layers contain a list of layer names that use the following format:

```
( ( "extractionName" ( "currentDensityName" "DFIIName" ) )
```

- `extractionName` is the layer name used in the DSPF or SPEF file.
- `currentDensityName` is the layer name used in the current density specification declared by the foundry, and it is also used as the current density specification in the EM rule file.
- `DFIIName` is the layer name used in the Cadence DFII technology file.

The extraction, current density, or DFII layer names used depends on how the rule files are written.

For example,

```
xrefLayers = (
( "poly1con" ("cw" "cw"))
( "poly" ("poly1" "poly1")))
```

The DFII names are used in textual reports as well as the GUI.

Sample EM Data File

```
(
; this file is case sensitive

routingLayers = ("Poly" "Metal1" "Metal2" "Metal3" "Metal4" "Metal5" "Metal6")

viaLayers = ("Cont" "Nimp" "Pimp" "Via1" "Via2" "Via3" "Via4" "Via5")

viaWidthList = (("Nimp" 0.2) ("Pimp" 0.2) ("Cont" 0.2) ("Via1" 0.2)
("Via2" 0.2) ("Via3" 0.2) ("Via4" 0.2) ("Via4" 0.2))

xrefLayers = (
( "POLYcont" ("Cont" "Cont"))
( "NSDcont" ("Nimp" "Nimp"))
( "PSDcont" ("Pimp" "Pimp"))
( "poly" ("Poly" "Poly"))
( "mt1" ("Metal1" "Metal1"))
( "mt2" ("Metal2" "Metal2"))
( "mt3" ("Metal3" "Metal3"))
( "mt4" ("Metal4" "Metal4"))
```

```
( "mt5" ("Metal5" "Metal5"))
( "mt6" ("Metal6" "Metal6"))
( "Via2NoCapInd" ("Via2" "Via2"))
( "Via2" ("Via2" "Via2"))
)

avgCurrentDensSpecList = (
(nil layer "Poly" minW 0.0 maxW -1.0 minL 0 maxL 5 currentDensity ((1.0 , 110)))
(nil layer "Metal1" minW 0.0 maxW -1.0 currentDensity ((1.1 , 110)))
(nil layer "Metal2" minW 0.0 maxW -1.0 currentDensity ((1.2 , 110)))
(nil layer "Metal3" minW 0.0 maxW -1.0 currentDensity ((1.3 , 110)))
(nil layer "Metal4" minW 0.0 maxW -1.0 currentDensity ((1.4 , 110)))
(nil layer "Metal5" minW 0.0 maxW -1.0 currentDensity ((1.5 , 110)))
(nil layer "Metal6" minW 0.0 maxW -1.0 currentDensity ((1.6 , 110)))
(nil layer "Cont" minW 0.0 maxW -1.0 res 1.0 currentDensity ((2.0 , 110)))
(nil layer "Nimp" minW 0.0 maxW -1.0 res 1.1 currentDensity ((2.1 , 110)))
(nil layer "Pimp" minW 0.0 maxW -1.0 res 1.2 currentDensity ((2.2 , 110)))
(nil layer "Via1" minW 0.0 maxW -1.0 res 1.3 currentDensity ((2.3 , 110)))
(nil layer "Via2" minW 0.0 maxW -1.0 res 1.4 currentDensity ((2.4 , 110)))
(nil layer "Via3" minW 0.0 maxW -1.0 res 1.5 currentDensity ((2.5 , 110)))
(nil layer "Via4" minW 0.0 maxW -1.0 res 1.6 currentDensity ((2.6 , 110)))
(nil layer "Via5" minW 0.0 maxW -1.0 res 1.7 currentDensity ((2.7 , 110)))
)
peakCurrentDensSpecList = (
(nil layer "Poly" minW 0.0 maxW -1.0 currentDensity ((1.1 , 110)))
(nil layer "Metal1" minW 0.0 maxW -1.0 currentDensity ((1.2 , 110)))
(nil layer "Metal2" minW 0.0 maxW -1.0 currentDensity ((1.3 , 110)))
(nil layer "Metal3" minW 0.0 maxW -1.0 currentDensity ((1.4 , 110)))
(nil layer "Metal4" minW 0.0 maxW -1.0 currentDensity ((1.5 , 110)))
(nil layer "Metal5" minW 0.0 maxW -1.0 currentDensity ((1.6 , 110)))
(nil layer "Metal6" minW 0.0 maxW -1.0 currentDensity ((1.7 , 110)))
(nil layer "Cont" minW 0.0 maxW -1.0 res 1.0 currentDensity ((2.1 , 110)))
(nil layer "Nimp" minW 0.0 maxW -1.0 res 1.1 currentDensity ((2.2 , 110)))
(nil layer "Pimp" minW 0.0 maxW -1.0 res 1.2 currentDensity ((2.3 , 110)))
(nil layer "Via1" minW 0.0 maxW -1.0 res 1.3 currentDensity ((2.4 , 110)))
(nil layer "Via2" minW 0.0 maxW -1.0 res 1.4 currentDensity ((2.5 , 110)))
(nil layer "Via3" minW 0.0 maxW -1.0 res 1.5 currentDensity ((2.6 , 110)))
(nil layer "Via4" minW 0.0 maxW -1.0 res 1.6 currentDensity ((2.7 , 110)))
(nil layer "Via5" minW 0.0 maxW -1.0 res 1.7 currentDensity ((2.8 , 110)))
)
```

10

Static Power Grid Calculator

This chapter describes how to analyze the effects of parasitics on power net wiring, without performing a full simulation, using the Virtuoso® UltraSim™ static power grid calculator.

Analyzing Parasitic Effects on Power Net Wiring

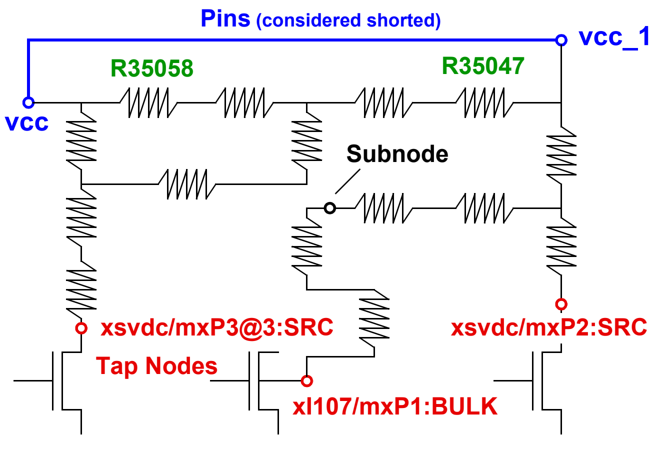
The static power grid calculator can be used to calculate all pin-to-tap or pin-to-subnode resistances based on the net description from a DSPF or SPEF file (a pre-layout netlist file is not required). The analysis assumes that all pins are connected, and an ordered list of instance pins or taps and their resistances is reported.

To invoke the static power grid calculator, use the `-r` command line option accompanied by a command file. The command file contains all of the calculator control and data filtering options. See Figure 10-1 on page 564 for more information about inputs to the static power grid calculator.

Figure 10-1 Static Power Grid Calculator Input

.

Design Content (DSPF, pll.spf)



*|NET vcc 0PF
*|P (vcc_1 B 0 -1.441 419.479)
*|P (vcc B 0 2158.41 419.491)
*|I (xl107/mxP1:BULK xl107/mxP1 BULK B 0 2128.46 392.112)
*|I (xsvdc/mxP3@3:SRC xsvdc/mxP3@3 SRC B 0 1837.78 431.24)
*|I (xsvdc/mxP2:SRC xsvdc/mxP2 SRC B 0 1837.78 429.16)
...
R35047 vcc_1 vcc_7 0.001 $l=0.004 $w=10 $lvl=195
R35058 vcc vcc_6 0.001 $l=0.004 $w=10 $lvl=195

Command File (spres.config)

- file pll.spf
- rmin 0.001
- net vcc
- sortby r

Calculator Command

```
ultrasim -r spres.config
```

Definitions

Power Net	Power supply net represented by RC network, defined in `*|NET`
Pin	Node of power net connected to power supply, defined in `*|P`
Tap Node	Node of power net connected to active device, defined in `*|I`
Subnode	Internal node of power network not connected to power supply or active device

The analysis results are printed into text files (two files per net) with filename prefixes specified in the command line option. The `-rout` filtering routine is used to filter existing output data without running the calculator again. See static power grid calculator output in Figure 10-2 on page 564.

Figure 10-2 Static Power Grid Calculator Output

.

Calculator Output (res-vcc.nr1000minr1.report)

Count	R	W/L	(x:y)	hname	name	(x:y)in GDSII
#1	1920.2579	5.00	(1440:366)	xl107/mxP1:BULK	mxP1	(1440:366)
#2	1577.1482	10.00	(1440:476)	xsvdc/mxP3@3	mxP3@3	(1440:476)
#3	1353.6123	4.00	(1446:509)	xsvdc/mxP2:SRC	mxP2:SRC	(1446:509)

ultrasim -r

Spectre Syntax

```
ultrasim –r <resistance_command_file> -o <output_prefix> [+log <log_file_name> |
     -log]
```

SPICE Syntax

```
.ultrasim –r <resistance_command_file> -o <output_prefix> [+log <log_file_name> |
     -log]
```

Arguments

`-r`	Enables static power grid calculator
`-o`	Output files prefix
`+log`	Prints log on display and into a specified log file
`-log`	Calculated resistances are not copied to a file

Command File Options (resistance_command_file)

`file` <*file_name*>	DSPF or SPEF filename.
`net` <*net_name*>	Defines power net for calculation. Net needs to be defined in `DSPF *|NET` section of file. Multiple net statements are allowed.
`addnetpin` <*net_name*> <*node_name*>	Converts subnode to pin in `net` <*net_name*>.
`netdeletepin` <*net_name*> <*node_name*> [<*node_name*>]	Converts pins to subnode.
`subnode` <*subnode_pattern*> [<*subnode_pattern*>]	Converts subnodes to tap nodes.
`netdeletetap` <*net_name*> <*node_name*> [<*node_name*>]	Convert tap nodes to subnodes.

`ipin <pin_pattern>`	Specifies tap nodes for resistor calculation. All other tap nodes are converted to subnodes.
`layer0ohm <layer_name> [<layer_name>]`	Resistors with specified layer names are shorted.
`layerfactor <layer_name> <scale_factor>`	Resistors with specified layer name are scaled by factor. Multiple `layerfactor` options are accepted.
`rmin <value>`	Resistors with a value less than `rmin` are shorted.
`sortby <sortkey>`	Specifies sorting method where `sortkey` can be `r`, `w/l`, or `rw/l`.
`report <report_options>`	Inserts `-rout` filtering options into command file (see `<report_options>` for definition).

Filtering Routine

The `-rout` command line option allows you to filter results from the static power grid calculator analysis without having to rerun the calculation. The calculator reads the existing `-r` output files and applies filtering to the data stored in these files.

Spectre Syntax

```
ultrasim –rout <prefix_of_rout_file> <report_options> [+log <log_file_name> |
     -log]
```

SPICE Syntax

```
.ultrasim –rout <prefix_of_rout_file> <report_options> [+log <log_file_name> |
-log]
```

Arguments

`-rout`	Enables filtering routine
`<prefix_of_rout_file>`	Defines output file prefix

<*report_options*>	Defines options filtering
+log	Prints log on display and into a specified log file
-log	Calculated resistances are not copied to a file

Arguments for Data Filtering (report_options)

-pat <*"pattern_with_wildcards"*>	Limits report to nodes matching specified pattern
-nr <*integer_value*>	Limits number of output nodes (no limit if not defined)
-minr <*double_value*>	Reports only nodes with `Reff>minr`
-maxr <*double_value*>	Reports only nodes with `Reff<maxr;`
-xmin <*value*> -xmax <*value*> -ymin <*value*> -ymax <*value*>	Reports only nodes in specified region
-gdsmag <*value*>	Specifies geometry magnification where all coordinates are multiplied by <*value*> if <*value*> is positive, or divided by <*value*> if <*value*> is negative
-gdsunits <*value*>	Specifies geometry units within the DSPF or SPEF file

For tap nodes connected to a MOSFET gate, w/l is reported to be zero. If the source and drain of the same MOSFET are connected to the same net, w/l for this tap node is also reported to be zero.

Examples

In the first static power grid calculator example, the following `Rescalc.config` command file is used:

```
file cell.spf
net gnd
netdeletepin gnd gnd_1
subnode  gnd:* vcc:*
```

```
rmin  0.01
sortby w/l
report –nr 1000 –minr 20 -pat*.SRC -pat*.DRN
net vcc
netdeletetap vcc x1/m2:drain x1/m4:source
addnetpin vcc vcca:1
```

The static power grid calculator is run using the following options:

```
ultrasim –r Rescalc.config –o res_examp
```

The static power grid calculator results are printed into the following files:

```
- res_examp-gnd.rout (non-filtered results)
- res_examp-gnd.minr20nr1000.rout (filtered results)
- res_examp-vcc.rout (non-filtered results)
- res_examp-vcc.minr20nr1000.rout (filtered results)
```

In the next example,

```
ultrasim –rout res_examp-gnd –o -nr 1000 -minr 5 -maxr 1000
```

a filtering routine is used on the `res_examp-gnd.rout` output file to filter out the first 1000 resistive paths with a resistance between 5 and 1000 Ohm. The filtered results are printed into the `res_examp-gnd.nr1000minr5maxr1000.report` file.

11

Fast Envelope Simulation for RF Circuits

This chapter describes fast envelope simulation for RF circuits, a simulation technique developed to reduce the large number of time steps and high computational costs associated with conventional transient analysis.

The Virtuoso® UltraSim™ simulator uses the harmonic balance technique, Fourier collocation, and pseudo-spectral method for fast envelope simulation. Fast envelope simulation is most efficient for RF circuits with a modulation bandwidth that is orders of magnitude lower than the clock frequency. For example, circuits with a clock that is the only fast varying signal, and with other input signals that have a spectrum with a frequency range orders of magnitude lower than the clock frequency.

In general, fast envelope simulation is not intended for circuits working with multiple carriers (fundamentals). It can be used for specific classes of circuits that operate with multiple, proportionate fundamentals. In this case, the greatest common denominator for all fundamental frequencies should be used as the clock frequency.

The Virtuoso UltraSim simulator supports the following types of fast envelope simulation: Normal, frequency modulation, and autonomous. Fast envelope simulation can be performed locally for a subcircuit or globally for an entire circuit.

Simulation Parameters

Fast envelope simulation with the Virtuoso UltraSim simulator uses the same setup as transient (`.tran`) simulation. Table 11-1 on page 570 lists all of the fast envelope simulation parameters. The parameters are defined in the `usim_opt` statement (refer to "Netlist File Formats" on page 51 for more information about this statement).

Note: You can use the `-h` option to print a help message for the designated simulation parameter (for example, `ultrasim -h env_method` displays help information for the `env_method` parameter).

Table 11-1 Fast Envelope Simulation Parameters

Parameter	Description
`env_clockf`	Specifies the carrier or clock for fast envelope simulation. Once `env_clockf` is specified, and its value is greater than 0, the Virtuoso UltraSim simulator performs a fast envelope simulation (if parameter is not specified, or the value is not greater than 0, a transient analysis is performed). **Note:** The `env_clockf` parameter is required for fast envelope simulation.
`env_method`	Specifies the fast envelope simulation method used by the simulator. ■ **hb** harmonic balance technique, simulating all sample points simultaneously while skipping as many cycles as possible (default). ■ **fc_full** full solve technique, based on Fourier collocation method and simulating all sample points simultaneously while skipping as many cycles as possible (similar to `hb` simulation method). ■ **fc_env** envelope solve technique, based on Fourier collocation method and simulating one point for each cycle without skipping a cycle. If the envelope solve simulation fails, the Virtuoso UltraSim simulator automatically performs a transient analysis. ■ **fc_env_full** envelope and full solve techniques, based on Fourier collocation method. The simulator performs an envelope solve simulation as often as possible without skipping a cycle (if an envelope solve simulation fails, the simulator performs a full solve simulation). ■ **fc_adapative** adaptive solve technique, based on Fourier collocation method. A partial solve simulation is performed between the envelope and the full solve simulations (no cycles skipped).

Table 11-1 Fast Envelope Simulation Parameters, *continued*

Parameter	Description
`env_nsamples`	Specifies the number of sample (Fourier collocation) points in one carrier or clock period for fast envelope simulation. The default is nine sample points. The `env_nsamples` value must be an odd-numbered integer greater than or equal to 3, otherwise the simulator sets the value of the parameter to the most suitable number. In cases where the greatest common denominator of multiple carrier frequencies is used as the clock frequency, `env_nsamples` needs to be large enough to assure that there are a minimum of three sample points for each carrier. For example, if there are two carriers with f1=1GHz and f2=3GHz, `env_clockf` needs to be set to 1GHz and `env_nsamples` to nine sample points or greater, so a minimum of three points are sampled in one period for the f2=3GHz carrier.
`env_maxnstep`	Specifies the maximum number of cycles that can be skipped for the `hb` and `fc_full` simulation methods, and the maximum number of steps for an envelope solve simulation with one point for each clock period (default is 10 cycles). The greater the value of `env_maxnstep`, the higher the speed of the fast envelope simulation over the transient analysis, and the lower the accuracy. To maintain a balance between speed and accuracy, set the parameter to large values when the carrier frequency is much greater than the frequency of the base band signal.
`env_tstart`	Specifies the start time for fast envelope simulation (default is three clock periods). For circuits with waveforms that have significant transient changes at the beginning of the analysis, set `env_tstart` to a time point after the transient changes have subsided.
`env_tstop`	Specifies the stop time for fast envelope simulation (value specified in `.tran` statement is default). The `env_tstop` parameter is useful for fast envelope simulation of a specific time interval and subsequent transient analysis of the remaining simulation time.

Table 11-1 Fast Envelope Simulation Parameters, *continued*

Parameter	Description
`env_tol`	Controls the accuracy of envelope solve for fast envelope simulation (default is set using `env_speed`). The range of values for `env_tol` is between 0 and 1. A value closer to 0 increases accuracy, but decreases speed with respect to transient analysis.
`env_trtol`	Multiplies `env_tol` for local truncation error (LTE) checking of envelope solve in fast envelope simulation (use `env_speed` to set default). A value closer to 1 decreases accuracy, but increases speed with respect to transient analysis.
`env_speed`	Specifies the speed setting for fast envelope simulation (settings are 1-5 and `env_speed=5` is the default value). A value closer to 1 increases accuracy, but decreases speed with respect to transient analysis. The `env_speed` settings include: ■ env_speed = 1 (env_tol = 0.0001; env_trtol = 40) ■ env_speed = 2 (env_tol = 0.001; env_trtol = 30) ■ env_speed = 3 (env_tol = 0.01; env_trtol = 20) ■ env_speed = 4 (env_tol = 0.1; env_trtol = 10) ■ env_speed = 5 (env_tol = 1; env_trtol = 5) The fast envelope simulation settings are more aggressive than the ones used in transient analysis due to the assumption that signal envelope is smooth and can be approximated by a low order polynomial.

Table 11-1 Fast Envelope Simulation Parameters, *continued*

Parameter	Description
`env_harms`	Specifies the number of harmonics for the carrier frequency with which time varying spectra are calculated and saved. The value of `env_harms` needs to be larger than or equal to 0 and less than or equal to `env_nsamples/2`. The default is 0 because the fast envelope simulation is designed to speed up transient analysis for modulated circuits. If you are interested in the time varying spectra or the adjacent channel power ratio (ACPR) results, `env_harms` needs to be set to a positive non-zero value (`env_harms=1` works for most applications). **Note:** Avoid setting `env_harms` to a high value because it can significantly increase computation time and reduce the speed of the fast envelope simulation.
`env_resolve`	Defines the flag used to enable resolve during simulation time synchronization. If `env_resolve=0`, resolve is disabled (default). When resolve is enabled (`env_resolve=1`), the fast envelope simulation resolves the system using a transient analysis when time synchronization has failed. Otherwise, fast envelope simulation performs interpolation during time synchronization. Enabling resolve decreases simulation speed, yet increases simulation accuracy.

Table 11-1 Fast Envelope Simulation Parameters, *continued*

Parameter	Description
`env_ignore_digital`	Defines the flag used to allow fast envelope simulation to ignore the timing of digital clocks during simulation. If `env_ignore_digital=1`, the digital clock timing is ignored (default). If the circuit consists of digital clocks with frequencies that are in the same order as the carrier frequency (`env_clockf`), you do not need to run fast envelope simulation if the clocks are not ignored because the simulation is slower than a transient analysis, due to the frequent resolves. Generally, these clocks do not affect the envelopes of the output signals in the RF portion of the circuit, and can be ignored. If the clock frequencies are significantly larger than the carrier frequency, and the frequencies need to be considered in the fast envelope simulation, set `env_ignore_digital=0` to consider digital clock timing and obtain accurate results.
`env_sim_mode`	Simulation mode for envelope simulation (default value is `normal`). ■ **env_sim_mode=normal** specifies a normal envelope simulation. ■ **env_sim_mode=fm** specifies a frequency modulation envelope (FM) simulation. ■ **env_sim_mode=osc** specifies an autonomous envelope simulation. A FM source must be present in the circuit for FM envelope simulation. The Virtuoso UltraSim simulator performs a FM envelope simulation according to the source and uses FM envelope simulation techniques. If a FM source is not in the circuit, the FM envelope simulation is disabled by the simulator, even if `env_sim_mode` is set to `FM`. **Note:** A reference port must be specified for autonomous envelope simulation using the `env_ref_port` option.

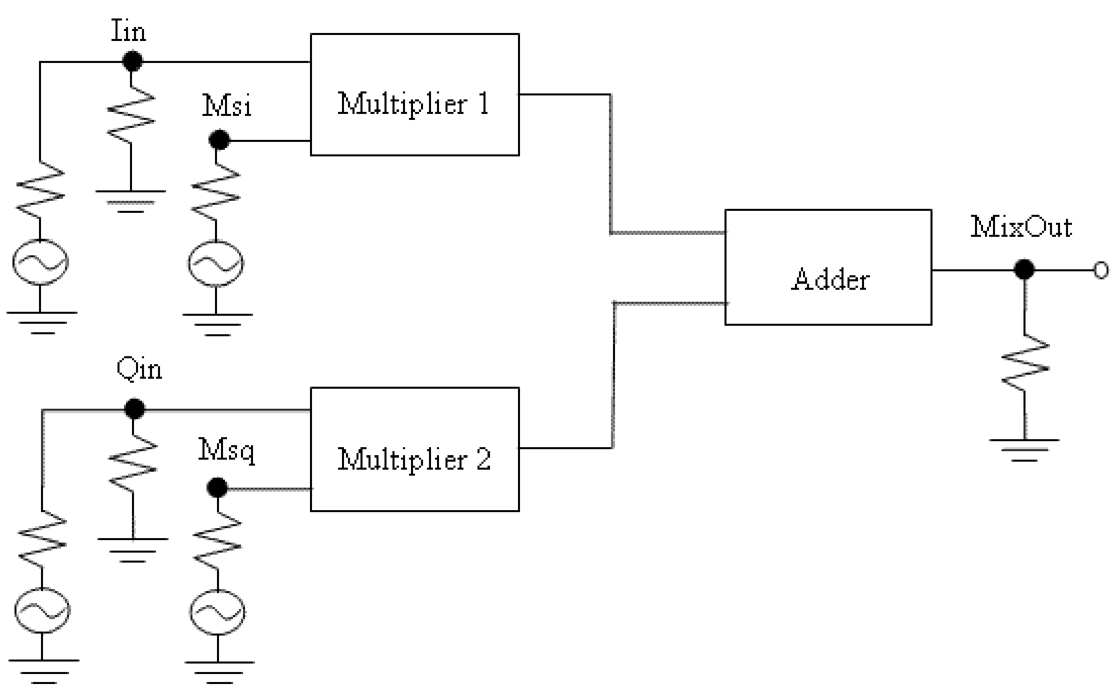
Table 11-1 Fast Envelope Simulation Parameters, *continued*

Parameter	Description
`env_ref_port`	Specifies the reference port for autonomous envelope simulation. ■ **env_ref_port=["N+ N-"]** `N+` and `N-` are the positive and negative nodes of the reference port, respectively. **Note:** The `env_ref_port` option is required for autonomous envelope simulation.

Example

The circuit in this fast envelope simulation example is the multiplier and adder portion of a code division multiple access (CDMA) circuit. The circuit consists of two multipliers and a single adder (see Figure 11-1 on page 575).

Figure 11-1 CDMA Circuit with Multipliers and Adder



The `I` channel signal and a sinusoidal source or carrier with a 90 degree phase shift are input into `Multiplier 1`. The `Q` channel signal and a sinusoidal source are input into

`Multiplier 2`. The two modulated signals output from `Multiplier 1` and `Multiplier 2` are combined in the `Adder`. The `Adder` output is sent to the CDMA circuit amplifier (not considered in this example). The multipliers and adder are modelled by two Verilog-A modules. The `I` and `Q` channel signals are input with two data files, and the carrier frequency is 1 Ghz.

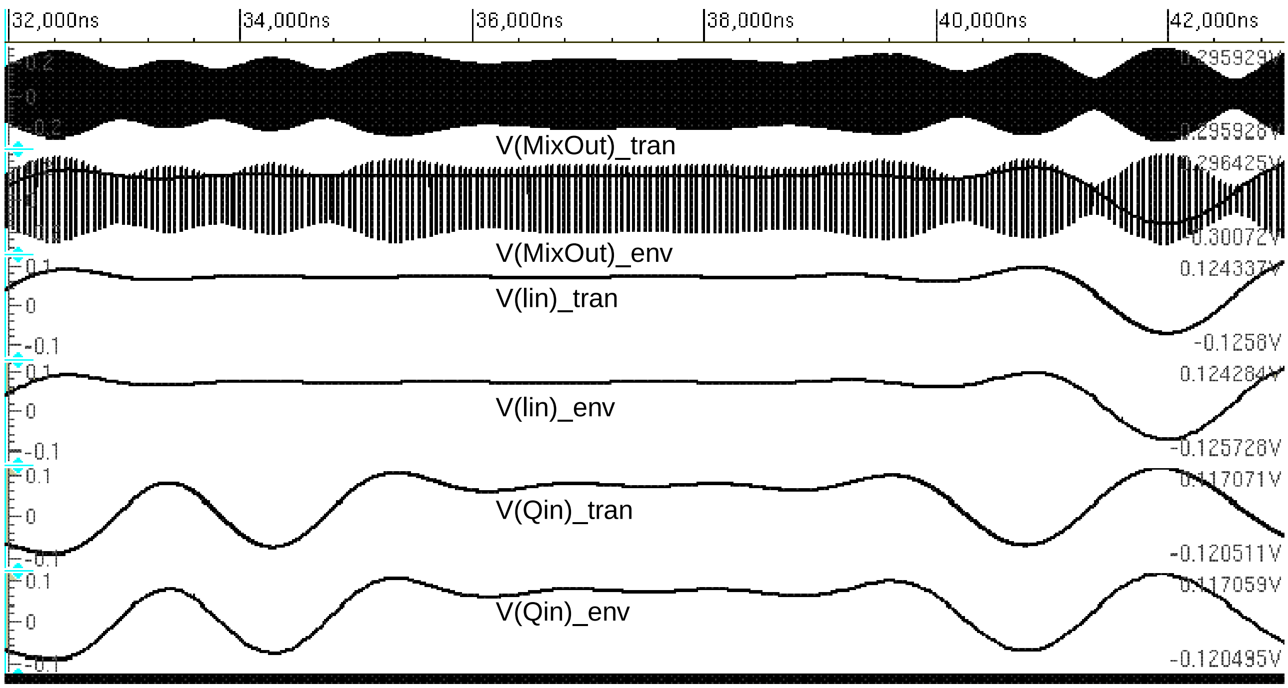
The netlist file for the sample CDMA circuit is shown below:

```
// Example for fast envelope simulation
// The multiplier and adder portion of a CDMA circuit
simulator lang=spectre
global 0
include "quantity.spectre"
parameters plo=-10  fcar=1G
Rout (MixOut 0) resistor r=50
Riin (Iin    0) resistor r=50
Rqin (Qin    0) resistor r=50
Im1  (Msi Iin M1out) multiplier
Im2  (Msq Qin M2out) multiplier
Ia   (M1out M2out MixOut) adder
PORT_Iin (Iin 0) port r=50 type=pwl phase=0 pwldbm=plo fundname="Iin" \
                 file="i_data.ascsig"
PORT_Qin (Qin 0) port r=50 type=pwl phase=0 pwldbm=plo  fundname="Qin" \
                 file="q_data.ascsig"
PORT_Msq (Msq 0) port r=50 type=sine freq=fcar ampl=1 fundname="fcar"
PORT_Msi (Msi 0) port r=50 type=sine freq=fcar ampl=1 sinephase=90 \
                 fundname="fcar"
saveOptions options save=allpub
ahdl_include "adder_veriloga.va"
ahdl_include "multiplier_veriloga.va"
simulator lang=spice
.tran 0.1u 100u
.probe tran v(*)
.measure tran VMixOutmax max 'v(MixOut)' from=10u to=100u
.measure tran VMixOutmin min 'v(MixOut)' from=10u to=100u
.usim_opt sim_mode=s
.usim_opt env_clockF=1G env_maxnstep=50 env_method=hb
.end
```

The transient analysis for this circuit takes 88 seconds, whereas the fast envelope simulation takes only 2.2 seconds, a 40 time increase in simulation speed.

The waveforms for the fast envelope simulation and transient analysis are plotted in Figure 11-2 on page 577. From `V(MixOut)`, it is clear that many cycles are skipped during the fast envelope simulation, resulting in a significant increase in simulation speed. The `I` and `Q` signals are generally the same when comparing the transient analysis to the fast envelope simulation.

Figure 11-2 Fast Envelope and Transient Analysis Waveforms



Local Envelope Simulation

All of the options listed in Table 11-1 on page 570 can be followed by a local scope, or subcircuit primitives and instances, to enable Virtuoso UltraSim local envelope simulation. The syntax used to define local envelope simulation at the subcircuit level is the same as described in the “Syntax” section of “Setting Virtuoso UltraSim Simulator Options” on page 155.

For example,

Spectre Syntax:

```
usim_opt env_clockf=1e6 env_method=HB env_maxnstep=50 inst=[x1.x2 x5]
```

SPICE Syntax:

```
.usim_opt env_clockf=1e6 env_method=HB env_maxnstep=50 inst=[x1.x2 x5]
```

tells the Virtuoso UltraSim simulator to run a local envelope simulation on instances `x1.x2` and `x5` with a clock frequency of `1e6`.

In the next example,

Spectre Syntax:

```
usim_opt env_clockf=1e6 env_method=HB env_maxnstep=50 subckt=[Ckt1]
```

SPICE Syntax:

```
.usim_opt env_clockf=1e6 env_method=HB env_maxnstep=50 subckt=[Ckt1]
```

tells the Virtuoso UltraSim simulator to run a local envelope simulation on all instances calling subcircuit `Ckt1` with a clock frequency of `1e6`.

For local envelope simulation, the lower level subcircuit follows the envelope simulation options defined at the higher level, if the value of the lower level `env_clockf` option is not larger than zero.

Note: A local envelope simulation is not defined for this subcircuit.

If the values of `env_clockf` are greater than zero at both the lower and higher levels, the envelope simulation options are reconciled according to the following rules.

Assuming there are two options defined (`Option1` and `Option2`), a new option (`Option3`) that is consistent with the first two options is created using these rules:

```
Option3.env_clockf = min(Option1.env_clockf, Option2.env_clockf)
Option3.env_nsamples = max(Option1.env_nsamples, Option2.env_nsamples)
Option3.env_maxnstep = max(Option1.env_maxnstep, Option2.env_maxnstep)
Option3.env_speed = min(Option1.env_speed, Option2.env_speed)
Option3.env_tol = min(Option1.env_tol, Option2.env_tol)
Option3.env_trtol = min(Option1.env_trtol, Option2.env_trtol)
Option3.env_tstart = max(Option1.env_tstart, Option2.env_tstart)
Option3.env_tstop = min(Option1.env_tstop, Option2.env_tstop)
Option3.env_harms = max(Option1.env_harms, Option2.env_harms)
Option3.env_resolve = max(Option1.env_resolve, Option2.env_resolve)
Option3.env_method = max(Option1.env_method, Option2.env_method)
    with FC_ENV < FC_ADAPTIVE < FC_ENV_FULL < FC_FULL < HB
```

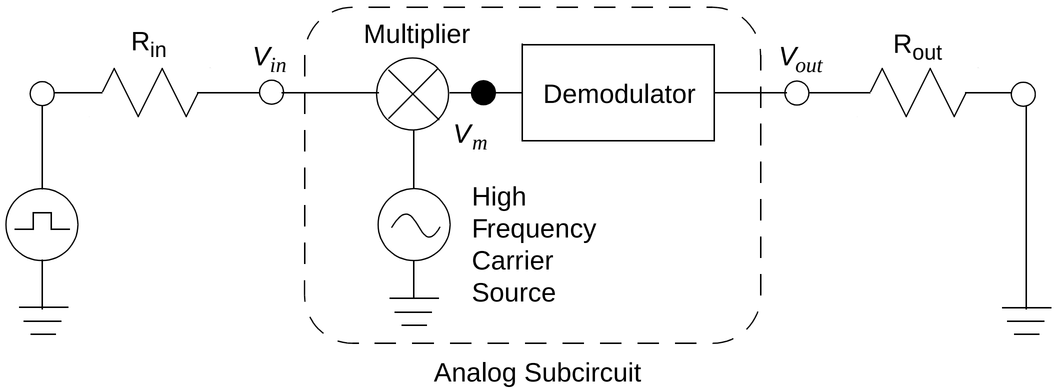
where `min` and `max` represent the minimum and maximum functions.

The subcircuits defined for local envelope simulation flatten up to the highest level through the hierarchy. Two hierarchically independent subcircuits defined for local envelope simulation may also be placed into the same partition for envelope simulation using the reconciled options, if the subcircuits are strongly coupled, this is automatically determined by the Virtuoso UltraSim partition algorithm.

Example

The circuit shown in this example (see Figure 11-1 on page 575) is a simple digital modulation circuit. The digital input is represented by a pulse source. The subcircuit for the analog section consists of a multiplier, a high frequency carrier source, and a demodulator. The multiplier and demodulator are implemented by Verilog-A models.

Figure 11-3 Digital Modulation Circuit



The netlist file for the sample digital modulation circuit is shown below:

```
* Example for local envelope simulation
* A simple digital modulation and demodulation circuit

simulator lang=spectre
global 0

ahdl_include "demodulator.va"
ahdl_include "multiplier.va"

subckt Analog vin vout
I1 (Vsin vin vm) multiplier
I2 (vm vdif vout) demodulator w=6.28*1.0e9
V1 (Vsin 0) vsource type=sine freq=1.0e9 ampl=1 sinephase=0.0
ends Analog

Myopt options save=all
simulator lang=spice
```

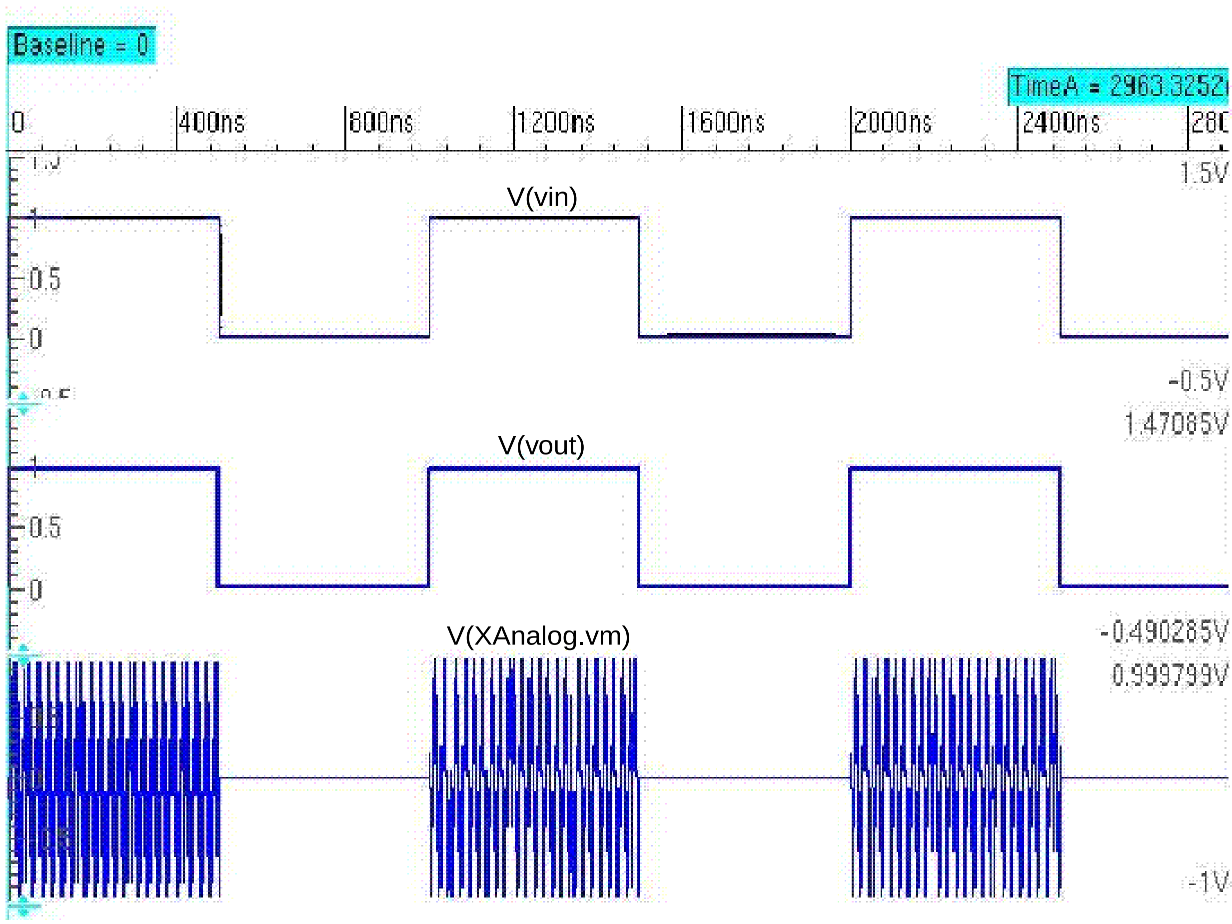
```
Vin vin1 0 pulse(0 1 1e-9 1e-11 1e-11 0.5e-6 1e-6)
XAnalog vin vout Analog
Rin  vin1 vin 1
Rout vout 0 1

.tran 0.1e-6 3e-6
.probe tran v(*)
.usim_opt sim_mode=s env_clockf=1e9 env_maxnstep=1000 env_method=hb inst=[XAnalog]

.end
```

The Virtuoso UltraSim simulator performs the local envelope simulation for the `XAnalog` analog subcircuit instance, as specified in the `usim_opt` statement. The local envelope simulation is completed in approximately one second. The waveforms for the simulation are plotted in Figure 11-4 on page 581.

Figure 11-4 Local Envelope Simulation Waveforms



Frequency Modulation Envelope Simulation

Frequency modulation (FM) envelope simulation can be used to simulate RF circuits with FM signals specified in the modulation sources.

Frequency Modulation Source Types

The FM source types include:

- Ideal
- One Data File
- Two Data Files

Ideal

The ideal FM source is defined as

$$V(t) = A\sin(2\pi f_i t + \phi_0)$$

where *A* is the amplitude, ϕ_0 is the initial phase, and f_i is the instantaneous frequency defined by

$$f_i = f_c + k\sin(2\pi f_m t)$$

where f_c is the carrier frequency, *k* is the modulation index, and f_m is the modulation frequency.

One Data File

The FM source with one data file can be expressed by

$$V(t) = A\sin(2\pi f_c t + \theta(t)\phi_0)$$

where

$$\theta(t) = k\int_0^t \Delta\omega(\tau)d\tau$$

and where $\Delta\omega$(t) is the radial frequency change defined in the data file. The instantaneous frequency can be calculated by

$$f_i = f_c + \frac{k\Delta\omega(t)}{2\pi}$$

Two Data Files

The FM source with two data files, one for `I` data and one for `Q` data, can be expressed by

$$V(t) = A(t)\sin(2\pi f_c t + \theta(t) + \phi_0)$$

where

$$A(t) = \sqrt{[I(t)]^2 + [Q(t)]^2}$$

$$\theta(t) = k\tan^{-1}\left(\frac{Q(t)}{I(t)}\right)$$

The instantaneous frequency can be calculated using

$$f_i = f_c + \frac{1}{2\pi}\frac{d\theta(t)}{dt}$$

Frequency Modulation Source Parameters

The FM source parameters are listed in the following table.

Table 11-2 Frequency Modulation Envelope Simulation Parameters

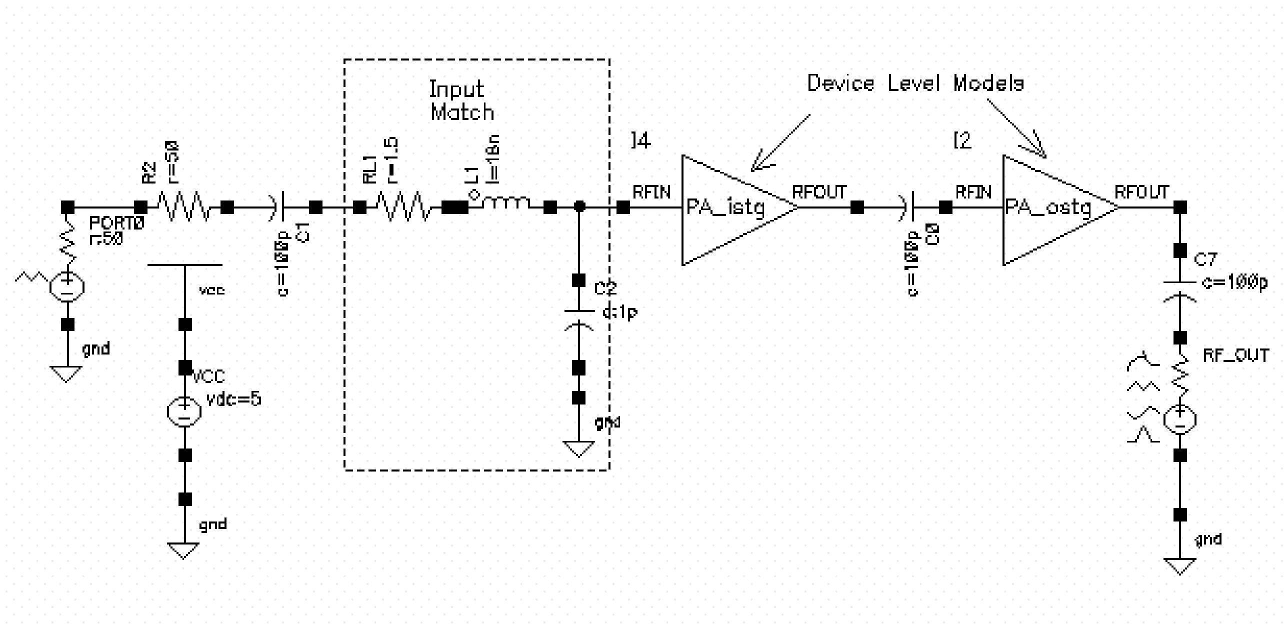
Parameter	Description
`fmmodindex`	Specifies the frequency modulation (FM) for the `k` index.
`fmmodfreq`	Specifies modulation frequency for the ideal FM sources.

`fmmodfiles`	Specifies the filenames for the FM sources with frequency modulation data. The format for this parameter is as follows.
	One File
	`fmmodfiles = [`*`data_file_name`*`]`
	Two Files
	`fmmodfiles = [`*`I_data_file_name Q_data_file_name`*`]`
	The data file format is the same as the piece-wise linear (PWL) file format (see Chapter 2, “Netlist File Formats” for more information about PWL format).

Example

A direct conversion transmitter circuit is used in the FM envelope simulation example (see Figure 11-5 on page 584).

Figure 11-5 Direct Conversion Transmitter Circuit



The netlist file for the sample direct conversion transmitter circuit is shown below.

```
// Direct conversion transmitter
// Example for FM envelope simulation

simulator lang=spectre
global 0 vcc!
include "rfModels.scs"

// EF_PA_ostg
subckt EF_PA_ostg RFIN RFOUT inh_bulk_n
    Q2 (net58 RFIN net92 inh_bulk_n) npnStd area=100
    Q1 (vcc! net104 net54 inh_bulk_n) npnStd area=10
    Q0 (net104 net108 net57 inh_bulk_n) npnStd area=5
    L6 (net43 RFIN) inductor l=13n
    L1 (vcc! RFOUT) inductor l=10n
    C20 (vcc! RFOUT) capacitor c=1p
    R52 (0 net54) resistor r=1K
    R60 (vcc! RFOUT) resistor r=1K
    R62 (RFOUT net58) resistor r=2
    R34 (0 net92) resistor r=10
    R53 (net108 net54) resistor r=400
    R33 (net54 net43) resistor r=20
    R51 (0 net57) resistor r=200
```

```
    I8 (vcc! net104) isource dc=3m type=dc
ends EF_PA_ostg

// EF_PA_istg
subckt EF_PA_istg RFIN RFOUT inh_bulk_n
    Q1 (vcc! net6 net18 inh_bulk_n) npnStd area=10
    Q2 (RFOUT RFIN net24 inh_bulk_n) npnStd area=25
    Q0 (net6 net8 net22 inh_bulk_n) npnStd area=5
    I8 (vcc! net6) isource dc=2m type=dc
    R34 (0 net24) resistor r=20
    R60 (vcc! RFOUT) resistor r=50
    R51 (0 net22) resistor r=200
    R33 (net18 net20) resistor r=200
    R53 (net8 net18) resistor r=1K
    R52 (0 net18) resistor r=1K
    L6 (net20 RFIN) inductor l=13n
ends EF_PA_istg

// Top circuit
I2 (net8 net9 0) EF_PA_ostg
I4 (net20 net1 0) EF_PA_istg
RF_OUT (RFOUT 0) port r=50 num=2 type=dc
PORT1 (net64 0) port r=50 num=1 type=sine freq=1e9 ampl=1 \
    fmmodfiles=["i_data.ascsig" "q_data.ascsig"] fmmodindex=1 fundname="carrier"
R2 (net64 net33) resistor r=50
RL1 (net24 net30) resistor r=1.5
R9 (net15 0) resistor r=50
R8 (net17 0) resistor r=50
L1 (net30 net20) inductor l=18n
VCC (vcc! 0) vsource dc=5 type=dc
C1 (net24 net33) capacitor c=100p
C0 (net8 net1) capacitor c=100p
C2 (net20 0) capacitor c=1p
C7 (RFOUT net9) capacitor c=100p

save RFOUT

tran tran stop=3e-6 start=0.0

simulator lang=spice
.usim_opt env_clockf=1e+09 env_fm_mod=1 env_method=fc_full env_maxnstep=100
.usim_opt wf_format=psf
```

```
.end
```

The FM source is defined in `PORT1`, with the `i_data.ascsig` and `q_data.ascsig` data files specifying the `I` and `Q` data, respectively.

A transient and fast envelope simulation was used to simulate the circuit. The transient simulation finished in 35.68 seconds and the fast envelope simulation in 3.89 seconds (fast envelope simulation is approximately 10 times faster than the transient simulation). The v(RFOUT) transient and fast envelope simulation waveforms are plotted in Figure 11-6 on page 586 and Figure 11-7 on page 587.

Figure 11-6 V(RFOUT) Transient Simulation Waveform

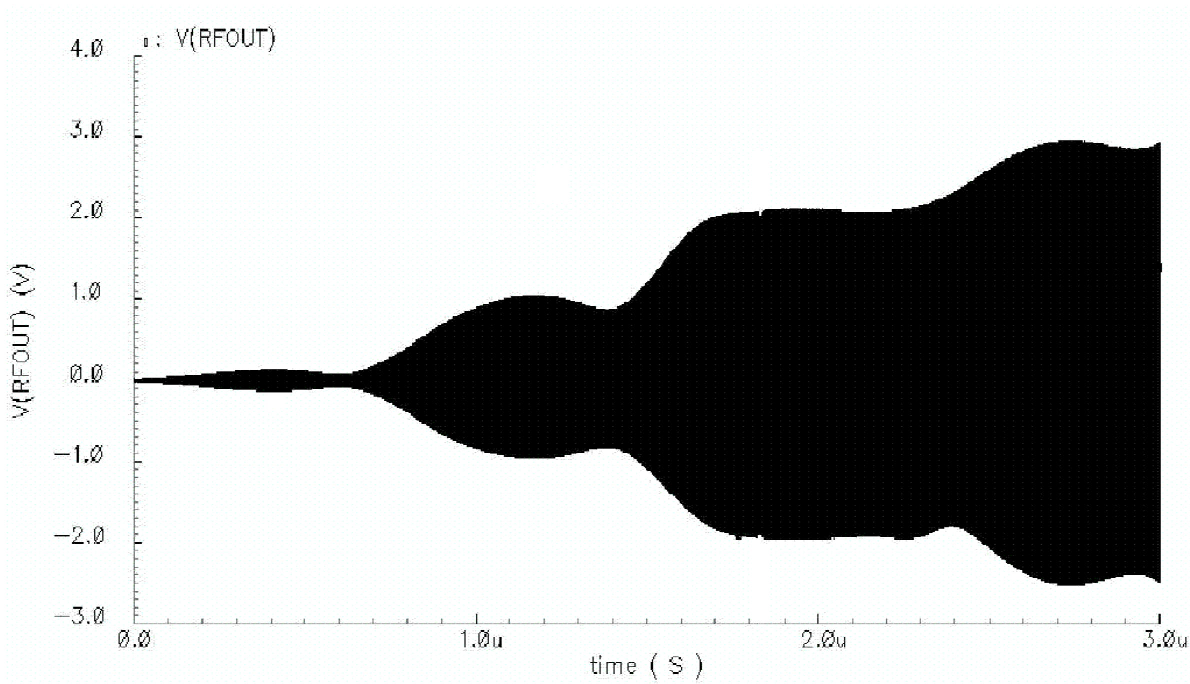
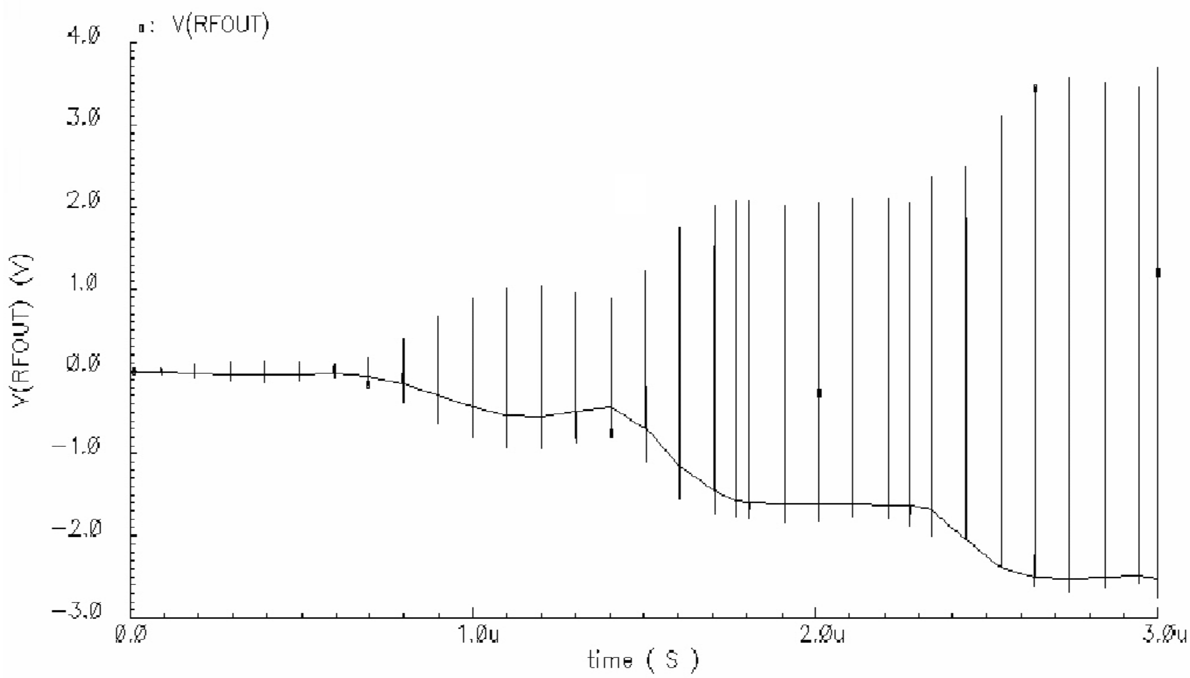


Figure 11-7 V(RFOUT) Fast Envelope Simulation Waveform



Autonomous Envelope Simulation

The Virtuoso UltraSim autonomous envelope simulation can be used to:

- Simulate RF circuits consisting of oscillation circuitries.
- Accelerate autonomous transient simulation.
- Simulate VCO/PLL with time varying control sources and oscillator-driven RF circuits.

To run an autonomous envelope simulation, set `env_sim_mode=OSC` in your netlist file. A reference port must be specified using the `env_ref_port` parameter. The value of `env_clockf` is used as the reference frequency for autonomous envelope simulation.

Tip

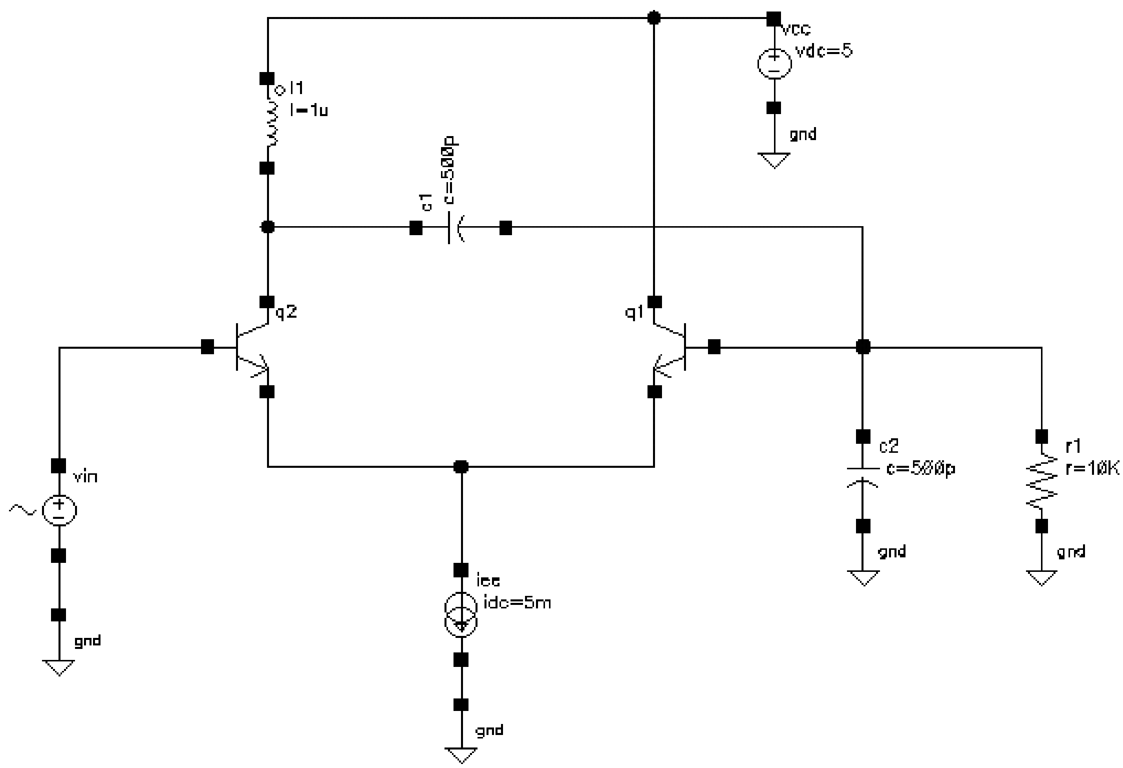
Set the `env_clockf` value close to the actual oscillation frequency for a faster and more accurate simulation.

Autonomous envelope simulation can be performed globally or locally. If `env_sim_mode=OSC` and `en_ref_port` are set for the entire circuit, global autonomous envelope simulation is performed. If `env_sim_mode=OSC` and `en_ref_port` are set for a subcircuit, local autonomous envelope simulation is performed.

Example

A BJT oscillator circuit, shown in Figure 11-8 on page 588, is used in this autonomous envelope simulation example.

Figure 11-8 BJT Oscillator Circuit



The netlist file for the sample BJT oscillator circuit is shown below.

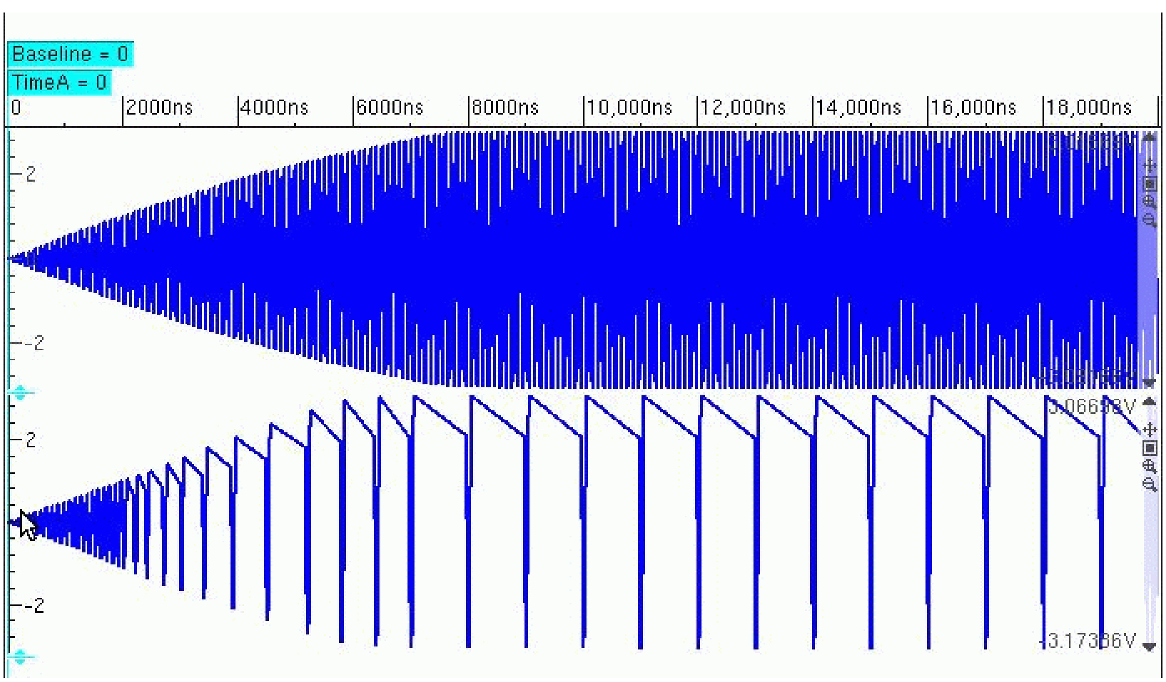
```
// BJT ecp oscillator
// Example for autonomous envelope simulation
simulator lang=spectre
save 1 2 3 4
simulator lang=spice
vin 4 0 sin(0 0.1 10meg 0 5e6) ac 1
vcc 9 0 5
iee 3 0 1mA
```

```
q1  9 1 3 qnl
q2  2 4 3 qnl
l1  9 2 1uH
c1  2 1 500pF
c2  1 0 500pF
r1  1 0 10k
.model qnl npn (bf=80 rb=100 ccs=2pf tf=0.3ns tr=6ns
+               cje=3pf cjc=2pf va=50)
.tran 5ns 20us
.usim_opt env_clockf=1e7 env_sim_mode=osc env_ref_port=[? 2 0 ?] env_tol=1e-2
.usim_opt sim_mode=s speed=1
.end
```

The circuit is simulated using a normal transient simulation and an autonomous envelope simulation. The transient simulation ends in 1.75 seconds and the autonomous envelope simulation in 0.15 seconds, a 12 time performance advantage for the autonomous envelope simulation.

The v(1) waveforms for both simulations are plotted in Figure 11-9 on page 589.

Figure 11-9 V(1) Transient and Autonomous Envelope Simulation Waveforms



The v(1) waveforms generated in the transient simulation are displayed in the upper plot and the autonomous envelope simulation v(1) waveforms are displayed in the lower plot.

12

Virtuoso UltraSim Reliability Simulation

As device sizes are reduced in scale, degradations caused by various mechanisms become more of a limiting factor in circuit design. The Virtuoso® UltraSim™ simulator provides full-chip, transistor-level reliability simulations and gives the designer real-time simulation capabilities. The following key reliability simulation features are supported:

- Hot carrier injection (HCI) and negative bias temperature instability (NBTI) simulations
- User-defined degradation models through the Virtuoso Unified Reliability Interface (URI)
- Aged model and AgeMOS reliability analysis methods
- Full-chip view of HCI and NBTI timing effects

The HCI, NBTI, aged model, and AgeMOS methods are discussed in the following sections. For more information about URI, refer to the *Virtuoso Unified Reliability Interface Reference*.

One important concept of the degradation model is *age*, which is an intermediate parameter that links the device degradation physical mechanism with the circuit reliability simulation. It quantifies the device degradation by unifying various bias conditions. Devices that are *fresh* have a zero age value while more degraded devices have larger age values.

Other important concepts are *age* (or degradation) model, *aged* (or degraded) model, and *aging* (or reliability) simulation:

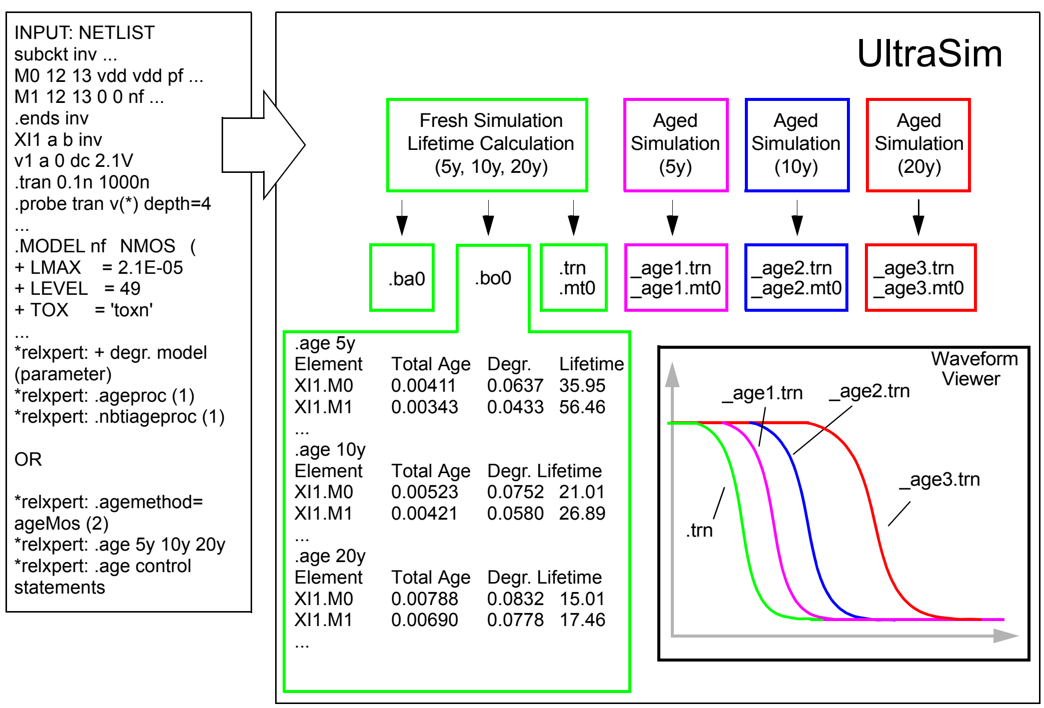
- **age model** expresses the physical mechanism of a certain degradation, such as HCI or NBTI
- **aged model** represents the effects of all kinds of degradations at a particular age value (that is, a degraded model card has an age model parameter called `age`, in addition to the SPICE parameters)
- **aging simulation** performs a whole circuit calculation, such as time analysis, with particular aged models

Figure 12-1 on page 592 shows the reliability simulation flow. The input is the SPICE netlist file, in addition to reliability control statements, degradation parameters, and one of the following reliability model options: Aged SPICE or AgeMOS model parameters.

The Virtuoso UltraSim simulator simulates the whole circuit, starting with the fresh model, and then calculates the age of each individual device in the circuit at each stress time. The reliability information, such as degradation and lifetime, is output into .bo0 and .ba0 files.

Note: The .bo0 file contains the total degradation of each device for all the age levels. The degradation for separated age level of each device is included in the result file `netlist_0.`*`level_number`* (where, *`level_number`* is the age level number). For example, the degradation of age level 0 for the netlist test.sp is included in test_0.level0, and age level 1 is included in test_0.level1.

Figure 12-1 Virtuoso UltraSim Reliability Simulation Flow



Aged model parameters are generated for degraded devices with the age value and a reliability model option. Reliability simulations are performed using these degraded models. Fresh and degraded waveforms are output to files during each simulation. By comparing the waveforms, you can determine how the degradations affect circuit performance (for example, timing).

Ultrasim URI supports two flows, namely analytical flow and table model flow. The analytical flow is more accurate compared to the table model flow and is compatible with the RelXpert simulator. The default flow of UltraSim URI is the analytical flow. Use the `age_analytical` UltraSim option in the netilst or the `ultrasim.cfg` file in the home directory to choose the flow of your choice:

Use the following setting to select the table model flow:

```
.usim_opt age_analytical=0
```

Note: When using the table model flow, ensure that you set the following environment variable:

```
setenv RX_OLD_URI 1 (The default value of RX_OLD_URI is 0)
```

Use the following setting to select the analytical flow:

```
.usim_opt age_analytical=1
```

Note: It is recommended that you use the analytical flow because it is compatible with the RelXpert simulator. In addition, use s mode (add .usim_opt sim_mode=s in the netlist) when running UltraSim simulator because the age simulation is highly dependent on the waveform.

Hot Carrier Injection Models

HCI degradation occurs when the channel electrons are accelerated in the high electric field region near the drain of the metal oxide semiconductor field-effect transistor (MOSFET) device and create interface states, electrons traps, or hole traps in the gate oxide near the drain. Drain current reduction, small signal performance deterioration, and threshold voltage shift are the typical forms of degradation that are detrimental to normal circuit function.

With designs moving into deep-submicron (DSM) levels, shorter channel lengths cause the electric field in the channel to become larger. Using the device-centric lightly doped drain (LDD) structure to alleviate HCI damage lowers the device current driving capability, and consequently degrades circuit performance. Trade-offs between HCI design rules and performance become increasingly complex as technology moves into smaller DSM levels. These conservative HCI design rules are a roadblock for high-performance design.

The MOSFET HCI model includes the following sub-models:

- Model for calculating substrate current [negative (NMOS) and positive-channel metal oxide semiconductor (PMOS)] and gate current (PMOS).
- Lifetime model which is used to calculate the HCI lifetime under circuit operating conditions.
- Aging model which describes the degradation of transistor characteristics as a function of stress.

  This model is used to generate degraded model parameters for aging simulation.

MOSFET Substrate and Gate Current Model

HCI degradation in n-channel MOSFETs is correlated to substrate current I_{sub}. The correlation exists because hot carriers and substrate current are driven by a common maximum channel electric field E_m factor, which occurs at the drain end of the channel. In p-channel MOSFETs, where the dominant driving force for degradation is charge trapping in the gate oxide, the degradation is found to be correlated to gate current I_g.

Hot Carrier Lifetime and Aging Model

This section describes the model used to predict HCI degradation from the substrate or gate current. The equation for degradation under DC stress conditions is first discussed. The model is then extended to AC bias conditions using quasi-static approximation. The Age parameter is used to quantify the amount of stress. It serves as a basis for determining HCI degradation under AC bias conditions from degradation under DC bias conditions.

HCI device degradation in a metal oxide semiconductor (MOS) is usually measured by the change in transconductance $\Delta g_m / g_m$, drain current $\Delta I_d / I_d$, and threshold voltage shift ΔV_t. Here, we generalize the degradation by using the ΔD symbol. The ΔD symbol can be replaced by any of the above quantities or other transistor parametric shifts in the following equations.

DC Lifetime and Aging Model

For the MOSFET under DC stress conditions, the amount of degradation is usually a function of time:

(12-1) $\Delta D = f(At)$

In general, the proportionality constant A describes the age (or degradation) rate as a function of channel electric field E_m and device bias condition

(12-2)

$$A = f(Vgs, Vds, Vbs, E_m)$$

AC Lifetime and Aging Model

Under the DC condition, Age is calculated using

(12-3) $Age(t) \equiv At$

Age is used to quantify the amount of hot carrier stress.

The amount of degradation ΔD is then

(12-4) $\Delta D(t) = f(Age)$

Using a quasi-static argument, under an AC bias condition, the Age definition is modified as follows

(12-5)

$$Age \equiv \int_0^T A dt$$

Using Equation 12-4 on page 595 and Equation 12-5 on page 595, you can determine the amount of degradation under the AC bias condition after a given time t or determine the AC lifetime τ.

Negative Bias Temperature Instability Model

A high vertical electrical field at a high temperature for TOX (MOSFET gate oxide thickness) 50 angstroms in length causes NBTI and makes the circuit fail immediately. The major damaging mechanism is the hole trapping and interface state generation. NBTI has become a major concern for reliable integrated complementary metal oxide semiconductor (CMOS) devices because of the threshold voltage (V_{th}) shift of p-MOSFET, Idsat reduction, and 1/f noise. Unlike HCI, NBTI can be a significant issue even when the drain-source is zero biased.

NBTI simulation is similar to HCI simulation with different lifetime parameters and degraded model sets for NBTI. If NBTI lifetime parameters are specified in the fresh model card, NBTI effects are simulated. NBTI and HCI effects can be simulated together or independently.

To simulate NBTI, the following is needed:

- NBTI lifetime model parameters specified in the fresh model card
- For aged model method, NBTI degraded SPICE model cards
- For AgeMOS method, AgeMOS parameters for NBTI in the fresh model card

Aged Model

The aged model is an extension of the traditional SPICE model for HCI, NBTI, or other age models. Aged SPICE model parameters are extracted from a fresh device at a number of stress intervals. These model parameters form a set of aged model files. Each file represents the transistor behavior after certain degradation, such as hot carrier stress. The amount of stress is given by the Age parameter calculated using Equation 12-5 on page 595. During the fresh simulation, the Virtuoso UltraSim simulator calculates the Age for each individual device. Using Age as a basis, the Virtuoso UltraSim simulator can construct a degraded model for each device from the aged model files. It can do this by interpolation or regression from these files in the linear-log or log-log domain of the calculation. The aged model method of calculating aged SPICE model parameter is shown graphically in Figure 12-2 on page 597. The P_1, P_2, and P_3 values are the degraded model parameters in SPICE model files with Age_1, Age_2 and Age_3 respectively. The P_i and P_r values are the respective model parameters if interpolation or regression is selected. Cadence recommends using interpolation in the log-log domain (default method). If there is a sign change in the P parameter, linear-linear interpolation is recommended.

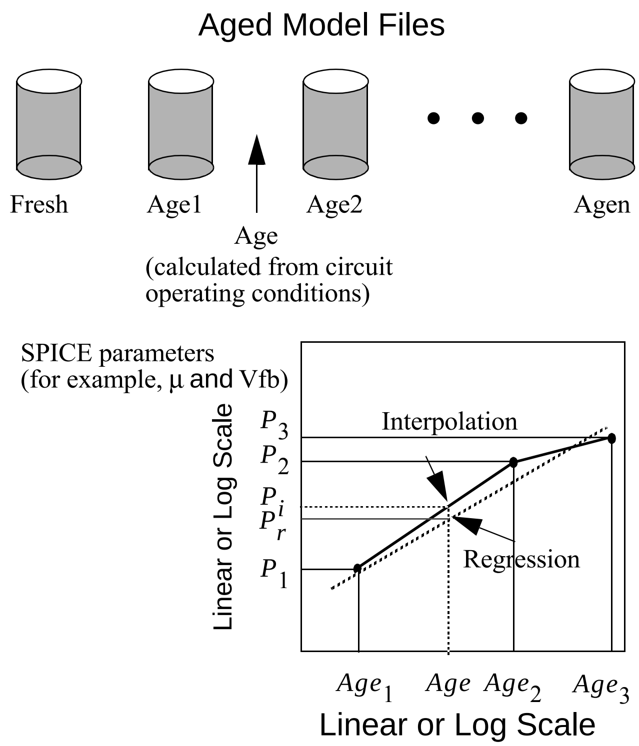
AgeMOS

The Cadence AgeMOS model provides a new reliability analysis method for HCI and NBTI circuit reliability simulation, especially for deep submicron CMOS reliability modeling and circuit simulation analysis. AgeMOS is applicable to any MOS SPICE model. The AgeMOS model is a significant improvement over other reliability models in the areas of model generation, accuracy, efficiency, and consistent circuit simulation.

Using this methodology, IC manufacturers can provide a universal model to all of their IC design customers without SPICE model compatibility issues. The AgeMOS model for HCI and NBTI enables designers to perform accurate and efficient reliability simulation analysis. This ensures optimal trade-off between yield and performance before product tape out. HCI and NBTI reliability analysis with the AgeMOS model prevents unnecessary reliability issues.

The Virtuoso UltraSim simulator accepts AgeMOS parameters for BSIM3V3 and BSIM4 models and supports the AgeMOS method for aged model card generation.

Figure 12-2 Calculating Aged Model Parameters



The degraded model parameter is a function of its fresh model and AgeMOS parameters. Calculate aged model parameters from aged model files using

(12-6) $$\Delta P = f(P0, age, d1, d2, n1, n2, s)$$

where Δ`P` is the change for the `P` parameter, `P0` is the fresh model parameter, `age` is the degradation age value, and `d1`, `d2`, `n1`, `n2`, and `s` are AgeMOS parameters.

The `h` prefix is used to specify the AgeMOS parameters for the HCI analysis. In the NBTI analysis, the AgeMOS parameters use the `n` prefix. The Virtuoso UltraSim simulator generates aged (or degraded) model cards in the circuit simulation using these AgeMOS parameters.

In the following HCI example,

```
*relxpert: +hd1_vth0 = 4.5 hd2_vth0 = 0 hn1_vth0 = 0.3 hn2_vth0 = 0.36488 hs_vth0
= 1.2777

*relxpert: +hd1_ua = 0.11812 hd2_ua = 13.12 hn1_ua = 0.2684 hn2_ua = 0.50428 hs_ua
= 3

*relxpert: +hd1_ub = 372.6 hd2_ub = 1 hn1_ub = 0.44 hn2_ub = 1 hs_ub = 1
```

```
*relxpert: +hd1_a0 = 0.40162 hd2_a0 = 0 hn1_a0 = 0.08392 hn2_a0 = 1 hs_a0 = 1
```

`vth0`, `ua`, `ub`, and `a0` changes with different age values. If `d1` and `d2` equal 0.0, the corresponding model parameter remains constant during the entire stressing. If `d1` and `d2` does not equal 0.0, the corresponding model parameter changes with stressing.

In order to specify age value for the aged model card, you need to add the age value to the fresh model card.

For example,

```
*relxpert: + age = 1e-12
```

Advantages of the AgeMOS Model

The AgeMOS model has the following advantages over the aged model:

- AgeMOS model is more accurate

  Aged parameters at any age value can be calculated using Equation 12-6 on page 597 (no interpolation or regression is needed).

- AgeMOS model keeps the aged parameters monotonic
- Simulation with the AgeMOS model is easier to perform

  Degraded model cards are not needed in the netlist file. Place AgeMOS parameters in the fresh model card along with the other age model parameters. The aged model parameters are calculated using the AgeMOS parameters.

- Simulation with the AgeMOS model is faster

  The aged model parameters are calculated directly with no interpolation or regression needed.

Reliability Control Statements

This section describes reliability control statements which are used to request an analysis, select a model, output control, or to pass other relevant information to the simulator.

Reliability control statements need to be included in the SPICE netlist file between the `.title` and `.end` cards. All control statements require `*relxpert`: at the beginning of the statement. The order of the statements in the netlist file is arbitrary. If the same control statement appears more than once, the statement that appears last overwrites all previous ones. A continuation line can be created by using `*relxpert: +` at the beginning of the line.

For more information about notations used to indicate how control statements are entered, see "Syntax" on page 29.

Previously, the UltraSim software supported the following reliability statement formats:

- `*relxpert: .age =1` (*rexlpert: is the prefix)
- `** .age=1` (** is the prefix)

Starting in the 7.1.1 release, the UltraSim parser supports only the first format for reliability statements. This means that only those reliability statements that have the `*relxpert:` prefix will be recognized, and the reliability statements with the `**` prefix will not be supported any more.

Reliability statement in Spectre format is as follows:

`*relxpert: age =1` (no period before the relxpert command) in Spectre format netlist

For example:

```
simulator lang = spectre
*relxpert: age 2min 20min 200min 400min
*relxpert: deltad 0.1
*relxpert: idmethod ids
*relxpert: vthmethod spice
*relxpert: agemethod agemos
simulator lang = spice
```

Reliability statement for SPICE format is as follows:

`*relxpert: .age=1` (there is a period before relxpert command) in the SPICE format netlist.

For example:

```
simulator lang = spice
*relxpert: .age 2min 20min 200min 400min
*relxpert: .deltad 0.1
*relxpert: .idmethod ids
*relxpert: .vthmethod spice
*relxpert: .agemethod agemos
simulator lang =spectre
```

The Virtuoso UltraSim simulator supports the following reliability control statements:

- .age on page 601
- .agemethod on page 602
- .ageproc on page 603
- .deltad on page 604
- .hci_only on page 605
- .minage on page 606
- .nbti_only on page 607
- .nbtiageproc on page 608

.age

```
*relxpert: .age time
```

Description

This statement specifies the future time (in seconds) at which the transistor degradation and degraded SPICE model parameters are calculated. The degraded SPICE model parameters are used in aged circuit simulation. This statement must be specified to invoke a reliability simulation. The calculated transistor degradation can be transconductance ($\Delta g_m / g_m$), linear or saturation drain current ($\Delta I_d / I_d$), degradation or threshold voltage shift (ΔV_t), or any other degradation monitor, dependent on how the lifetime parameters are extracted.

The default is MOS reliability simulation is not performed by the simulator.

Examples

```
*relxpert: .age 10y
*relxpert: .age 1y 2y 5y 8y 10y
```

.agemethod

`*relxpert: .agemethod { interp [ linlog|loglog ] }`

`*relxpert: .agemethod agemos`

Description

This statement specifies the method for calculating degraded SPICE model parameters for aging circuit simulation. The `interp` argument is used to select the method of interpolation for aged model files and `agemos` specifies which AgeMOS method is used. The domain (parameter versus Age) for performing the interpolation and regression can be linear-log or log-log. Cadence recommends using the interpolation in the log-log domain method.

The default is

```
*relxpert: .agemethod interp loglog
```

Examples

```
*relxpert: .agemethod interp loglog
*relxpert: .agemethod agemos
```

.ageproc

`*relxpert: .ageproc` *`mname`* `FILES =` *`fname1 fname2`* `[`*`fname3…`*`]`

Description

This statement specifies aged SPICE model files for generating HCI degraded SPICE models using the interpolation method (selected through .agemethod). The `mname` argument is the transistor model name that applies to the aged SPICE models and it must be the same model name used in the SPICE `.model` statement. The `fname1` argument specifies the model file containing the fresh model. All of the other model files `fname2`*…n* contain aged SPICE models. The order of the aged SPICE model files corresponds to increasing age values (that is, `fname1` is the fresh model file and *fnamen* is the aged model file with the highest age value).

Example

```
*relxpert: .ageproc nmos files=model/nmos0.mod
*relxpert: + model/nmos1.mod model/nmos2.mod
```

tells the Virtuoso UltraSim simulator any `mname` model without a corresponding `.ageproc` statement is not aged (that is, no degraded models are generated).

Note: Each `fname` file can only contain one `.model` statement.

.deltad

```
*relxpert: .deltad value
```

Description

This statement is used to perform the lifetime calculation for each transistor under circuit operating conditions. The criterion for lifetime is `value`. The degradation value can be transconductance ($\Delta g_m / g_m$), linear or saturation drain current degradation ($\Delta I_d / I_d$), or threshold voltage shift (ΔV_t), or any other degradation monitor, dependent on how the lifetime parameters are extracted.

Example

```
*relxpert: .deltad 0.1
```

tells the Virtuoso UltraSim simulator to perform a lifetime calculation for a 10% transconductance change in all devices ($\Delta D = 0.1$ for all devices).

.hci_only

```
*relxpert: .hci_only
```

Description

If this statement is specified, only HCI analysis is performed, even if NBTI models are included in the simulation.

.minage

```
*relxpert: .minage value
```

Description

If specified, this statement speeds up the aging calculation by using fresh SPICE model parameters if the transistor Age is smaller than `value` (set the smallest Age value for which degraded SPICE model parameters are calculated).

The default is

```
*relxpert: .minage 0.0
```

A degraded SPICE model is generated for transistor $Age > 0.0$.

Example

```
*relxpert: .minage 0.01
```

.nbti_only

```
*relxpert: .nbti_only
```

Description

If this statement is specified, only NBTI analysis is performed, even if HCI models are included in the simulation.

.nbtiageproc

```
*relxpert: .nbtiageproc mname files = fname1 fname2 [fname3...]
```

Description

This statement specifies aged SPICE model files for generating NBTI degraded SPICE models using the interpolation method (selected through .agemethod). The `mname` argument is the transistor model name that applies to the aged SPICE models and it must be the same model name used in the SPICE `.model` statement. The `fname1` argument specifies the model file containing the fresh model. All of the other model files `fname2`... *n* contain aged SPICE models. The order of the aged SPICE model files corresponds to increasing age values (that is, `fname1` is the fresh model file and *fnamen* is the aged model file with the highest age value).

Example

```
*relxpert: .nbtiageproc nmos files = model/nmos0.mod
*relxpert: + model/nmos1.mod model/nmos2.mod
```

tells the Virtuoso UltraSim simulator that any `mname` model without a corresponding `.nbtiageproc` statement is not aged (that is, no degraded models are generated).

Note: Each `fname` file can only contain one `.model` statement.

Virtuoso UltraSim Simulator Option

deg_mod

Spectre Syntax

```
usim_opt deg_mod={ e|r }
```

SPICE Syntax

```
.usim_opt deg_mod={ e|r }
```

Description

This option specifies the method used to calculate age rate (see Equation 12-2 on page 594).

The default is

```
.usim_opt deg_mod=r
```

Note: The equation-based calculation is denoted as `e` and the representative calculation is `r`.

Reliability Shared Library

uri_lib

SPICE Syntax

`.usim_opt uri_lib =` *`library_path`*

Or:

`*relxpert: .uri_lib` *`library_path`*

Description

This option loads a URI library. You can use both `.usim_opt` and `.relxpert:` syntax formats to set the `uri_lib` option for UltraSim reliability.

Examples

```
.usim_lib uri_lib = ./libURI.so
*relxpert: .uri_lib  ./libURI.so
```

Virtuoso UltraSim Simulator Output File

The Virtuoso UltraSim simulation results are stored in an output file ending with the suffix `.bo#`, where `#` is the alter number used in the netlist file.

The `bo#` file contains a list of all significantly degraded elements, as well as each elements name, total age, degradation, and lifetime:

- The calculated Age of the transistor (see Equation 12-5 on page 595).
- The transistor degradation after the time specified in the `.age` command.

  The degradation can be transconductance ($\Delta g_m/g_m$), linear or saturation drain current ($\Delta I_d/I_d$) degradation, threshold voltage shift (ΔV_t), or any other degradation monitor. The selection of this quantity depends on the type of degradation model parameters that are extracted.
- The lifetime of a transistor to reach the failure criterion specified in the `.deltad` command.

  The degradation can be transconductance ($\Delta g_m/g_m$), linear or saturation drain current ($\Delta I_d/I_d$) degradation, threshold voltage shift (ΔV_t), or any other degradation monitor. The selection of this quantity depends on the type of degradation model parameters that are extracted.

Example 1

```
*relxpert: .age 10Y
Elem name Total Age Degradation Lifetime


 XI0.M00.00399142   0.0832736   15.0193

 XI8.M00.00343223   0.0778054   17.4663

 XI4.M00.00342567   0.0777383   17.4998

 XI2.M10.00285925   0.177749    3.66879

 XI5.M10.00311931   0.181816    3.36506

 XI3.M10.003535     0.187481    2.97245

 XI0.M10.0038534    0.191414    2.7287
```

Example 2

Multiple stress time values:

```
*relxpert: .age 3.80518e-006Y
Elem name Total Age     Degradation Lifetime
M1    .27785e-0100.00384321 2.38626
.age 3.80518e-005Y
Elem name Total Age     Degradation Lifetime
M1    1.27785e-0090.00674256 2.38626
.age 0.000380518Y
Elem name Total Age     Degradation Lifetime
M1    1.27785e-0080.0118292 2.38626
```

The Age of all transistors is stored in a file with the suffix `.ba#`. The information stored in the file contains the following:

- The transistor name with subcircuit call name
- The Age in the forward and reverse modes of transistor operation

  The forward mode is defined when the degradation damage is found at the first (drain) node of the transistor.
- The total Age of the transistor without considering forward or reverse mode operation

Example 3

Multiple stress time values:

```
*relxpert: .age 0.000190259Y
Elem nameForward Age Reverse Age Total Age
XINV1.M12.67973e-0089.25076e-0102.77224e-008
XINV2.M12.62009e-0089.95977e-0102.71969e-008
.age 0.000570776Y
Elem name Forward Age Reverse Age Total Age
XINV1.M1 8.03919e-0082.77523e-0098.31671e-008
XINV2.M1 7.86028e-0082.98793e-0098.15907e-008
.age 0.000951294Y
Elem name Forward Age Reverse Age Total Age
XINV1.M1 1.33986e-0074.62538e-0091.38612e-007
XINV2.M1 1.31005e-0074.97989e-0091.35984e-007
```

Example 4

Multiple degradation models:

```
XI3.M0 0.00287476  0           0.00287476
XI2.M0 0.00256932  0           0.00256932
XIL1.M1 2.55524e-0060           2.55524e-006
    0.00214127  8.0686e-005 0.00222196
XIL2.M1 2.55524e-0060           2.55524e-006
    0.00214127  8.0686e-005 0.00222196
```

In this example, there are two lines for the `XIL1.M1` and `XIL2.M1` MOSFETs. Each line represents each degradation model for the MOSFETs. The order is the same as the order specified by the degradation models in the fresh model card.

13

Digital Vector File Format

This chapter describes how to perform vector checks and apply stimuli according to digital vectors using the Virtuoso® UltraSim™ simulator. To process digital vector file formats, the following statement needs to be specified in the netlist file:

Spectre Syntax

```
vec_include "vector_filename" [HLCheck=0|1] [autostop=true|false]
     [insensitive=yes|no]
```

SPICE Syntax

```
.vec "vector_filename" [HLCheck=0|1] [autostop=true|false] [insensitive=yes|no]
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.vec`).

Description

`HLCheck` is a special flag that you need to set to generate the vector output check for H and L states of input signals. Bidirectional and output signals always check H and L states and are unaffected by the `HLCheck` flag. Normally, you do not need to use the `HLCheck` flag unless it is necessary to check if input signals are shorted in the netlist file. The output resistance of H and L states for input signals can be specified by the `hlz` statement.

Each `vec` card can specify only one vector file. If a netlist file needs to include multiple vector files, multiple `vec` cards can be used. For example, if a netlist file needs to include three vector files, then it needs to use three `vec` cards.

Spectre Syntax:

```
Card 1: vec_include "file1.vec"
Card 2: vec_include "file2.vec"
Card 3: vec_include "file3.vec"
```

SPICE Syntax:

```
Card 1: .vec "file1.vec"
Card 2: .vec "file2.vec"
```

```
Card 3: .vec "file3.vec"
```

The Virtuoso UltraSim simulator handles the vector file content as case insensitive, except when called in Virtuoso Spectre® mode. For Spectre mode, use the `-spectre` option or input file name extension `*.scs`.

Arguments

`vector_filename`	The filename of the digital vector file
`HLCheck = 0 │ 1`	Special flag which turns on checking for the H and L states for input signals (default = 0)
`autostop=true│false`	■ **false** tells the Virtuoso UltraSim simulator to use the end time from the `.tran` or `tran` statement (default). ■ **true** tells the simulator to use the last specified time point in the vector file as the end time. If multiple `.vec` files are specified, and `autostop=true` is in one or all `.vec` statements, the simulator takes the longest time point available in the `.vec` files and uses it as the end time. **Note:** The `autostop` argument can also be used when loading `.vcd` and `.evcd` files. For more information about these files, refer to "Supported Netlist File Formats" on page 51.
`insensitive=yes│no`	Specifies whether vector file content is considered case sensitive or insensitive. By default, the Virtuoso UltraSim simulator determines case sensitivity based on the Spectre/SPICE mode used. When running in Spectre mode (`*.scs` file extension or `-spectre`), the vector file content is treated as case sensitive. In SPICE mode, the content is case insensitive.

Example

Spectre Syntax:

```
vec_include "vec1.vec" autostop=true
```

SPICE Syntax:

```
.vec "vec1.vec" autostop=true
```

tells the Virtuoso UltraSim simulator to replace the end time with the time from the `vec1.vec` file (that is, the time from the `vec1.vec` file is used as the transient simulation end time).

The digital vector file is described in detail in the following sections:

General Definition

Comment Line

A comment line begins with a semicolon ‘(`;`).

Note: A semicolon is only used in digital vector file format comment lines.

Continuous Line

A continuous line is indicated by a plus sign ‘(`+`).

The maximum length of a line is 1024 characters. If a card is longer than 1024 characters, you need to use the continuous line for the card.

Tip

For a long identifier (for example, a 1280-bit vector bus) that cannot fit on a single line, use the forward slash `\` sign after the last bit. Do not use a space between the last bit and the `\` sign. Put a space in front of the continuous vector or use a `+` sign. If you use a `+` sign, the continuous vector is treated as another vector bus.

Signal Mask

A signal mask can be used to specify the effective range of the current statement in a vector file (statement applies to specific signals). The Virtuoso UltraSim simulator matches the signals according to the signal definition order in the `radix`, `vname`, and `io` statements. For the corresponding signal, a value of 1 indicates the statement is valid and a value of 0 indicates the statement is ignored. Based on the size of the vector specified in the `radix` statement, the signal mask value can range from 0 to 1 for 1bit, 0 to 3 for 2bit, 0 to 7 for 3bit, and 0 to 9 or A to F for 4bit.

Example

```
radix 2 2 4
io i i o
vname A[1:0] B[1:0] P[3:0]
vih 2.5 3 0 0
vih 3.3 0 3 0
trise 0.5 1 2 0
chk_window -1 5 1 0 0 F
```

For more information about the statements used in this example, refer to "Vector Patterns" on page 617 and "Signal Characteristics" on page 631.

Vector Patterns

In this section, vector patterns (such as signal sizes, directions, names, and check windows) are defined. The Virtuoso UltraSim simulator supports the following digital vector pattern statements:

- radix on page 618
- io on page 619
- vname on page 620
- hier on page 622
- tunit on page 623
- chk_ignore on page 624
- chk_window on page 625
- enable on page 628
- period on page 630

radix

`radix` *`vector1_size1 vector2_size2 ...vector_sizeN`*

Description

Specifies the size (in bits) of the vector. This statement must be located before any other statements, and can only be specified once. Valid vector sizes include 1 (binary), 2, 3 (octal), or 4 (hexadecimal).

Tip

If the `radix` of the vector is larger than 1, the name of this vector specified in vname must be indexed as [msb:lsb] or [lsb:msb]. If the `radix` is 4, the `vname` can use names such as name[3:0] and name[0:3].

Examples

The following example

```
radix 2 2 4
```

contains three vectors: Two 2-bit vectors and one 4-bit vector.

Note: The examples presented in the rest of this chapter follow this format.

In the next example

```
radix 2 11 1111
```

also contains three vectors, two 2-bit vectors and one 4-bit vector, but in a different format.

io

`io` *`type1 type2 ...typeN`*

Description

The `io` statement defines the type of vector. It can be the `i` (input), `o` (output), or `b` (bidirectional) type. If this statement is specified more than once, the last value is used.

Notes

- Use the `enable` statement to specify the control signal for the bidirectional vector (`b`). If this specified control signal is not found, the Virtuoso UltraSim simulator issues an error.
- If the control signal of the bidirectional vector is not specified by the `enable` statement, the Virtuoso UltraSim simulator treats it as an input signal.

Example

```
radix 2 2 4
io i i o
```

The first and second vectors are input vectors, and the third vector is an output vector.

vname

vname name1 name2 ... nameN

Description

The `vname` statement assigns a name to each vector. For a single bit vector, it can have the following naming format: `Va`, `Va[0:0]`, or `Va[[0:0]]`. For multiple bit vectors, the naming formats include: `Va[2:0]`, `Va[[2:0]]`, `Va[0:2]`, or `Va[[0:2]]`. Each naming format is given a different resulting name. If this statement is specified more than once, the last value is used.

Hierarchical signal names are also supported by `vname`. That is, you can apply vector stimuli or perform a vector check on the internal signals of instances. When mapping hierarchical signal names, the default delimiter is a period (`.`). You can change the value of the delimiter using the `hier_delimiter` option in the analog netlist file. The hier statement can be used to enable or disable this option.

Table 13-1 vname Vector Names

Naming Format	Resulting Names
`Va[2:0]`	Va2, Va1, Va0
`Va[[2:0]]`	Va[2], Va[1], Va[0]
`Va[0:2]`	Va0, Va1, Va2
`Va[[0:2]]`	Va[0], Va[1], Va[2]
`X1.Va[0:2]`	Internal signals `Va0`, `Va1`, and `Va2` of instance `X1`
`TOP.X1.Va[[0:2]]`	Internal signals `Va[0]`, `Va[1]`, and `Va[2]` of instance `TOP.X1`

Tip

If the radix of the vector is larger than 1, the name of the vector specified in `vname` must be indexed as [msb:lsb] or [lsb:msb]. If `radix` is 4, `vname` can use names such as name[3:0] and name[0:3].

Examples

In the following example

```
radix 2 2 4
```

```
io i i i
vname va[1:0] vb[[1:0]] vc[[0:3]]
```

tells the Virtuoso UltraSim simulator that the voltage sources in the first vector are named va1 and va0. Voltage sources in the second vector are connected to vb[1] and vb[0]. The third vector has voltage sources with the names vc[0], vc[1], vc[2], and vc[3].

In the next example

```
radix 2 2 4
io i i o
vname X1.va[1:0] X2.vb[[1:0]] X1.X3.vc<[0:3]>
hier 1
```

tells the simulator the voltage sources in the first vector are mapped to internal signals va1 and va0 of instance X1. Voltage sources in the second vector are connected to v[1] and vb[0] of instance X2. The third vector defines the output vector check for signals vc<0>, vc<1>, vc<2>, and vc<3> of instance X1.X3.

hier

```
hier 0|1
```

Description

This option is used to specify whether or not the hierarchical signal name mapping feature is enabled. If `hier` is set to `0`, the hierarchical delimiter (for example, signal period or `.`) is considered to be part of the signal name. The default value is `1` (hierarchical signal name mapping enabled). If this statement is specified more than once, the last value is used.

Example

```
radix 2
io i
hier 0
vname X1.va[1:0]
```

tells the Virtuoso UltraSim simulator to connect the voltage sources with the `X1.va1` and `X1.va0` signals located in the top level of the analog netlist file.

tunit

```
tunit time_unit
```

Description

Sets the time unit for all time related variables. The time unit can be one of the following: `fs` (femto-second), `ps` (pico-second), `ns` (nano-second), `us` (micro-second), and `ms` (milli-second). The default time unit is 1 ns. If this statement is specified more than once, the last value is used.

Example

```
tunit 1.5ns
```

chk_ignore

```
chk_ignore start_time end_time [mask1 mask2 ... maskN]
```

Description

The `chk_ignore` statement specifies a window for ignoring output vector checks. A mask can be provided to specify which vector and bit to apply. If the mask is not specified, the setting applies to all output vectors. The `start_time` and `end_time` arguments must be specified. To define multiple time windows for ignoring output vector checks, use multiple `chk_ignore` statements.

Arguments

`start_time`	Defines the start time for the window used to ignore the output vector checks (use tunit to define the `start_time` units).
`end_time`	Defines the end time for the window used to ignore the output vector checks (use tunit to define the `end_time` units). You can use `end_time=-1` to ignore the entire transient time.

Example

```
tunit 1n
chk_ignore 0 100 0F30          ; 0F30 is a signal mask
chk_ignore 3e+2 500 0F30
chk_ignore 0 -1 F000           ; F000 is a signal mask
```

tells the Virtuoso UltraSim simulator to ignore the output vector check for signals specified by the mask `0F30` in the time windows 0 ns to 100 ns and 300 ns to 500 ns, and to ignore the entire transient time for the signals specified by the mask `F000`.

chk_window

```
chk_window start_time end_time steady [period=const [first=const] ] [mask1 mask2
     ... maskN]
```

Description

The `chk_window` statement specifies a window for vector checking. The Virtuoso UltraSim simulator only checks the signal states within this window. The signal states outside the window are ignored. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all output vectors. The checks occur at every time point specified in the vector file or as defined by the `period` and `first` arguments.

Setting the `period` argument activates periodic window checking. If `period` is not defined, the `first` argument is ignored by the simulator.

Notes

- To activate periodic window checking, you need to include the `"period="` and `"first="` keywords.
- Parameters and expressions are supported for the `start_time`, `end_time`, `period`, and `first` arguments (see “Examples” on page 626 for more information about parameters and expressions syntax).

Arguments

`start_time`	Defines the window start time at which the window starts at time `vec_time-start_time`. If the `period` argument is defined, `vec_time` is the first time point defined by the `first` argument, and the vector checks are repeated according to the value of `period`. If the `period` argument is not defined, `vec_time` is the time point defined in the vector file.
`end_time`	Defines the window end time at which the window ends at time `vec_time+end_time`.
`steady = 0 | 1`	Can be set to `0` or `1`. If set to `0`, then the vector check passes as long as the signal has reached the desired state once. If set to `1`, then the signal remains in the desired state for the entire window period to pass the vector check.
`period`	Activates periodic window checking and defines its time period.

`first`	Defines the first check point for periodic window checking (only valid when the `period` argument is also defined).

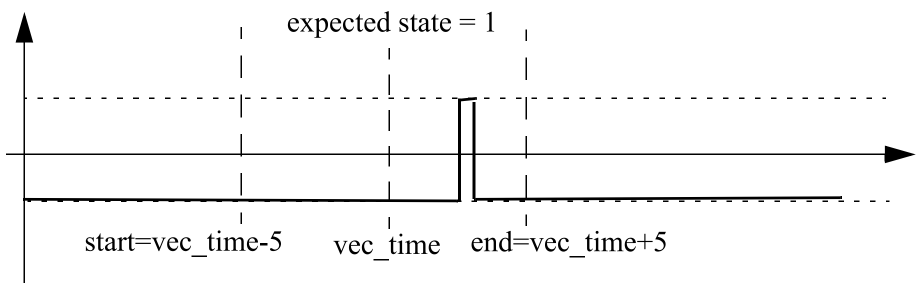
Examples

The following example

```
chk_window 5 5 0
```

tells the Virtuoso UltraSim simulator to set the steady state to `0`, so the waveform passes the vector check (see Figure 13-1 on page 626).

Figure 13-1 Vector Check with chk_window Steady State Set to 0

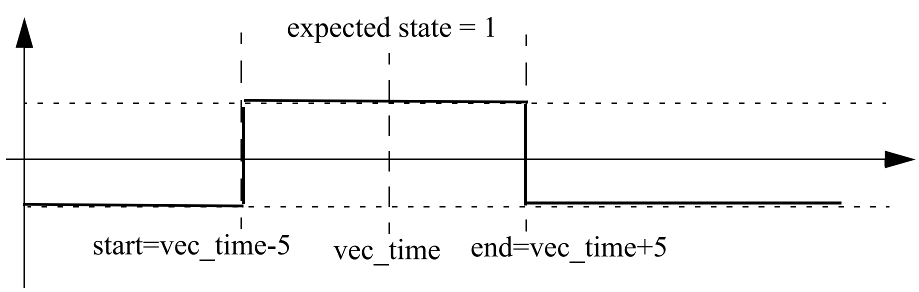


In the next example

```
chk_window 5 5 1
```

tells the simulator to set the steady state to `1`, which means the signal needs to stay at state `1` for the whole window period to pass the vector check, as shown in Figure 13-2 on page 626. If the signal is as shown in Figure 13-1 on page 626, the vector check fails.

Figure 13-2 Vector Check with chk_window Steady State Set to 1



In the next example

```
radix 1 1 1 1
```

```
vname ph1 d q qb
io i i o o
tunit 10ns
chk_window -10 30 1 period=100 first=5 0 0 1 0
```

tells the simulator to activate periodic window check for signal `q`. The vector check points start at 50 ns and repeat every 1 us.

In the next example

```
chk_window -10 30 1 first=5 0 0 1 0
```

tells the simulator to ignore the `first` argument because a valid `period` argument has not been specified.

In the next example

```
tunit 1ns
param myfadd(x,y)='x + y'
param mystartt=1.5 mystopt='(myfadd(mystartt, 50.5)'
chk_window mystartt mystopt 1
```

tells the simulator to set the steady state to 1, the start time for `chk_window` to 1.5 ns, and the end time to 52 n (this example shows the `chk_window` parameters and expressions syntax).

enable

```
enable 'enable_signal_expr' [mask1 mask2 ... maskN]
```

Description

The `enable` statement connects the enable signal, or enable signal expression, to the bidirectional vector. The resulting value 1 (H) enables the output signal. The controlled bidirectional signal is regarded as an input for other values.

You can provide a mask to specify to which vector and bit the enable signal expression applies. If the mask is not specified, the setting applies to all bidirectional vectors. Also, if this statement is specified more than once, the last value is used.

The enable signal can be used in a vector or an analog netlist file. When an enable signal is used in an analog netlist file, it can also be defined as an output signal for a vector check or only used as an enable signal. The avoh and avol statements can be used to define the logic high and low voltage thresholds for the analog signal.

Note: The enable signal cannot be defined as a bidirectional signal.

Bit-wise logic operators are supported in an enable signal expression: `&` (AND), `|` (OR), `^` (XOR), and `~` (NOT). Additional operators can be created using a combination of the supported operators. The order of processing for the logic operators is NOT > AND > OR, XOR (OR and XOR are processed at the same time). You can use parentheses `()` around the operators to change the processing order.

Note: You need to use single quotation marks `''` for enable signal expressions.

Examples

The following example

```
radix 1 1 1 1
io i i b o
vname en in bi out
enable en 0 0 1 0
```

tells the Virtuoso UltraSim simulator to set `en` as the enable signal for `bi`, and when `en` is in 1 (or H) state, `bi` becomes the output signal. When `en` is in 0 (or L, X, U) state, `bi` changes to the input signal. When `en` is in Z state, the `bi` (input and output) signal also changes to Z state.

In the next example

```
radix 1 1 1
io i b o
vname en bi out
enable ~en 0 1 0
```

tells the simulator to set `en` as the enable signal for `bi`. Unlike the first example, this enable signal name contains a ~ sign, which reverses the state to control the bidirectional signal. Now when the enable signal is in 1 (or H) state, the `bi` becomes an input signal.

In the next example

```
radix 1 1 1
io b b o
vname bi_1 bi_2 out
enable ana_en1 0 1 0
enable '(ana_en1 | X1.ana_en2) & out' 1 0 0
```

tells the Virtuoso UltraSim simulator that the `ana_en1` and `X1.ana_en2` enable signals originate in the analog netlist file, and `X1.ana_en2` is a hierarchical signal. Although the `out` signal is used as an enable signal, the simulator still performs a vector check.

period

`period` *`time`*

Description

The `period` statement is used to specify the time interval for tabular data, so that the absolute time is not needed.

If `period` is not specified, then the absolute time must be specified in the tabular data. If it is specified more than once, the last value is used.

Example

```
period 10.0
```

tells the Virtuoso UltraSim simulator that the signal period is 10 ns and the absolute time points are unnecessary.

Signal Characteristics

In this section, signal characteristics containing various attributes for input or output signals (such as delay, rise or fall time, voltage thresholds for logic low and high, and driving ability) are defined. For most of these statements, the mask can be used to apply the specified characteristics to the corresponding signals. The statements are organized into three groups:

- Timing on page 632
- Voltage Threshold on page 639
- Driving Ability on page 648

Note: In the following examples for time-related statements, the time unit is 1 ns if the statement is not specified with tunit.

Timing

Timing characteristics of input or output signals (such as delay, rise time, and fall time) can be specified using the following statements. The values of these statements can be positive or negative. For the delay timing characteristics, the negative value is used to advance the signals by a specified time. For the rise and fall timing characteristics, the negative value is the same as the positive one.

- idelay on page 633
- odelay on page 634
- tdelay on page 635
- slope on page 636
- tfall on page 637
- trise on page 638

Note: The Virtuoso UltraSim simulator checks whether the values of the `trise`, `tfall`, and `slope` statements are reasonable (warning message is issued when the defined value is too small or large).

idelay

`idelay` *`time_value [mask1 ... maskN]`*

Description

The `idelay` statement specifies the delay time for the corresponding input signal. If a bidirectional signal is specified, this applies only to the input stage of the bidirectional signal. The default value is 0.0, if `idelay` or `tdelay` is not set.

Example

```
idelay 5.0
```

tells the Virtuoso UltraSim simulator to delay all input signals by 5 ns, whereas

```
idelay -5.0
```

tells the simulator to advance all input signals by 5 ns.

odelay

`odelay` *`time_value [mask1 ... maskN]`*

Description

The `odelay` statement specifies the time delay for the corresponding output signal. If a bidirectional signal is specified, this applies only to the output stage of the bidirectional signal. The default value is 0.0, if `odelay` or `tdelay` is not set.

Example

```
odelay 5
```

tells the Virtuoso UltraSim simulator to delay all output signals by 5 ns, whereas

```
odelay -5.0
```

tells the simulator to advance all output signals by 5 ns.

tdelay

`tdelay` *`time [mask1 mask2 ... maskN]`*

Description

The `tdelay` statement specifies the delay time for corresponding vectors. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all vectors (input, output, and bidirectional).

If `tdelay` is not specified, the default value is 0.0. If this statement is specified more than once, the last value is used for the active mask. This statement can also overrule the value previously set by the `idelay` or `odelay` statements.

Examples

```
tdelay 5.0
```

tells the Virtuoso UltraSim simulator to advance all signals by 5 ns.

```
tdelay -5.5  3 0 F
```

tells the simulator to advance all signals, specified with a mask, by 5.5 ns.

slope

`slope` *`time [mask1 mask2 ... maskN]`*

Description

The `slope` statement sets the input vectors rise and fall time. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If this statement is not specified, then the default value of 0.1 ns is used. If this statement is specified more than once, the last value is used for the active mask. This statement can also overrule the value previously set by the `trise` or `tfall` statements.

Examples

```
slope 0.05
```

or

```
vname va[1:0] vb[[1:0]] vc[[0:3]]
io i i o
slope .025 1 3 5
```

The least significant bit, `va0`, of the first input vector and the two bits, `vb1` and `vb0`, of the second input vector have a `trise` and `tfall` of 0.025 ns. The third vector is an output vector (specified in the `io` statement), so it is not affected by the `slope` statement.

tfall

`tfall` *`time [mask1 mask2 ...maskN]`*

Description

The `tfall` statement specifies the falling time of the input vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

The value from the `slope` statement is used, if `tfall` is not specified. If this statement is specified more than once, the last value is used for the active mask. This statement can also overrule the value previously set by the `slope` statement.

Examples

The following example

```
tfall 0.05
```

tells the Virtuoso UltraSim simulator that all input vectors have a fall time of 0.05 ns.

In the next example

```
vname va[1:0] vb[1:0] vc[0:3]
tfall 0.1  0 2 0
```

the most significant bit, `vb[1]`, of the second input vector has a fall time of 0.1 ns. The fall time of `vb[0]` and other input vectors remains the same.

trise

trise time [mask1 mask2 ...maskN]

Description

The `trise` statement specifies the rise time of the input vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `trise` is not specified, the value from the `slope` statement is used. If this statement is specified more than once, the last value is used for the active mask. This statement can also overrule the value previously set by the `slope` statement.

Examples

The following example

```
trise 0.1
```

or

```
trise -0.1
```

tells the Virtuoso UltraSim simulator that all input vectors have a rise time of 0.1 ns.

In the next example

```
vname va[1:0] vb[1:0] vc[0:3]
trise 0.1  0 3 0
```

the two bits of the second input vector has a rise time of 0.1 ns and the `trise` of the other input vector remains the same.

Voltage Threshold

When converting input vectors to stimuli or performing an output vector check, the voltage threshold for logic low and high can be specified using the following statements:

- vih on page 640
- vil on page 641
- voh on page 642
- vol on page 643
- avoh on page 644
- avol on page 645
- vref on page 646
- vth on page 647

vih

`vih` *`voltage [mask1 mask2 ...maskN]`*

Description

The `vih` statement specifies the logic high voltage of the input vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `vih` is not specified, the default voltage is 3.3. If this statement is specified more than once, the last value is used for the active mask.

Examples

```
vih 5.0
```

or

```
vih 5.5  3 1 0
```

vil

`vil` *`voltage [mask1 mask2 ... maskN]`*

Description

The `vil` statement specifies the logic low voltage of the input vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `vil` is not specified, the default voltage is 0.0. If this statement is specified more than once, the last value is used for the active mask.

Examples

```
vil 0.25
```

or

```
vil 0.5  3 0 0
```

voh

`voh` *`voltage [mask1 mask2 ... maskN]`*

Description

The `voh` statement specifies the logic high voltage of the output vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all output vectors.

If `voh` is not specified, the default voltage is 3.3. If this statement is specified more than once, the last value is used for the active mask.

Examples

```
voh 5.0
```

or

```
voh 5.5  0 0 F
```

vol

```
vol voltage [mask1 mask2 ... maskN]
```

Description

The `vol` statement specifies the logic low voltage of the output vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all output vectors.

If `vol` is not specified, the default voltage is 0.0. If this statement is specified more than once, the last value is used for the active mask.

Example

```
vol = 0.05
voh = 1
```

tells the Virtuoso UltraSim simulator to interpret all output signals with values below 0.05 V as 0, print all signals above 1 V as 1, and all signals between 0.05 V and 1 V are U.

avoh

```
avoh voltage [ signal_name1 signal_name2 ... signal_nameN ]
```

Description

The `avoh` statement specifies the logic high voltage of the signal from the analog netlist file, which is not defined in the radix, vname, or io statements. You can provide signal names to specify the valid scope for `avoh` (wildcards are supported). A period (`.`) can be used as the hierarchical delimiter to specify the hierarchical signal. If a signal name is not used, the setting applies to all analog signals used in the vector file.

For more information about wildcards, see "Wildcard Rules" on page 55.

Note: A mask cannot be used to specify which vector and bit to apply to the signal (different behavior from other vector format statements).

Example

```
avoh = 1 ana_en* X1.Enanble
```

tells the Virtuoso UltraSim simulator that analog signals `ana_en*` and `X1.Enanble` have a logic high voltage of 1.0.

avol

`avol voltage [ `*`signal_name1 signal_name2`*` ... `*`signal_nameN`*` ]`

Description

The `avol` statement specifies the logic low voltage of the signal from the analog netlist file, which is not defined in the radix, vname or io statements. You can provide signal names to specify the valid scope for `avol` (wildcards are supported). A period (`.`) can be used as the hierarchical delimiter to specify the hierarchical signal. If a signal name is not used, the setting applies to all analog signals used in the vector file.

For more information about wildcards, see "Wildcard Rules" on page 55.

Note: A mask cannot be used to specify which vector and bit to apply to the signal (different behavior from other vector format statements).

Example

```
avol = 0.5 ana_en* X1.Enanble
```

tells the Virtuoso UltraSim simulator that analog signals `ana_en*` and `X1.Enanble` have a logic low voltage of 0.5.

vref

```
vref node_name [mask1 mask2 ... maskN]
```

Description

The `vref` statement sets the reference node of the input vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `vref` is not specified, the default value is 0 (that is, the ground). If this statement is specified more than once, the last value is used for the active mask.

Examples

The following example

```
vref 0
```

tells the Virtuoso UltraSim simulator to set the negative node of the vector source to ground.

In the next example

```
vref vss
```

tells the simulator to set the negative node of the vector source to `vss`.

Note: The Virtuoso UltraSim simulator only supports reference node to ground. References to other nodes causes the simulator to issue error messages.

vth

```
vth voltage [mask1 mask2 ... maskN]
```

Description

The `vth` statement sets the threshold voltage of the output vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all output vectors.

If `vth` is not specified, the default value is 1.65. If this statement is specified more than once, the last value is used for the active mask.

Examples

```
vth 2.5
```

or

```
vth 2.7 0 0 8
```

Driving Ability

For input stimuli, the output resistance of vector sources can affect Virtuoso UltraSim simulation results. To specify the driving ability of vector sources, use the following statements:

- hlz on page 649
- outz on page 650
- triz on page 651

hlz

`hlz` *`resistance [mask1 mask2 ... maskN]`*

Description

The `hlz` statement specifies the output resistance for the corresponding input vector, but unlike `outz`, this output resistance only applies to the H and L states of the vector. This resistance overwrites the resistance for the H and L states set by `outz`.

You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `hlz` is not specified, the default value follows `outz`. If `hlz` is set to 0, the Virtuoso UltraSim simulator uses 0.01 instead. If this statement is specified more than once, the last value is used for the active mask.

Examples

```
hlz 1meg
```

or

```
hlz 4.7k 2 2 0
```

outz

`outz` *`resistance`* `[`*`mask1 mask2 ... maskN`*`]`

Description

The `outz` statement specifies the output resistance for the corresponding input vector. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `outz` is not specified, the default value is 0.01. If `outz` is set to 0, the default value is used. If this statement is specified more than once, the last value is used for the active mask.

Examples

```
outz 1meg
```

or

```
outz 5.5meg 2 2 0
```

triz

`triz` *`resistance [mask1 mask2 ... maskN]`*

Description

The `triz` statement specifies the output impedance when the corresponding input vectors are in tri-state. You can provide a mask to specify which vector and bit to apply. If the mask is not specified, the setting applies to all input vectors.

If `triz` is not specified, the default value is 1,000 Meg. If `triz` is set to 0, the Virtuoso UltraSim simulator uses 0.01 instead. Also, if this statement is specified more than once, the last value is used for the active mask.

Examples

```
triz 2000meg
```

or

```
triz 550meg 2 2 0
```

Tabular Data

This section describes the values of signals at specified times (absolute or period time modes). For periodic signals, it is unnecessary to specify the absolute time at each time point. The `period` statement can be used to specify the signal period.

Absolute Time Mode

The period is not specified.

```
Time1 vector1_value1 vector2_value1 vector3_value1
Time2 vector1_value2 vector2_value2 vector3_value2
...
TimeN vector1_valueN vector2_valueN vector3_valueN
```

Period Time Mode

The period is specified.

```
vector1_value1 vector2_value1 vector3_value1
vector1_value2 vector2_value2 vector3_value2
...
vector1_valueN vector2_valueN vector3_valueN
```

vector_value can be 0-9, A-F, Z, X, L, H, or U, and is dependent on how `radix` is set.

Description

Tabular data is used to describe the waveform of voltage sources.

Examples

```
; format: time vector
0 000101010
10 011010101
20 000101010
```

or

```
; format: vector
00101010
11010101
00101010
```

Note: This example assumes the period has been set by `period` 10.0.

or

```
; format: time vector
10 02A
20 315
30 02A
```

Valid Values

The valid values in tabular data depend on the `radix` statement setting.

Table 13-2 Tabular Data Valid Values

Value Specified in `radix` Statement	Valid Value
`1`	0, 1
`2`	0-3
`3`	0-7
`4`	0-9, A-F

The values specified in the table above are converted into 0 and 1 states by the Virtuoso UltraSim simulator. The simulator also accepts `L`, `H`, `Z`, `X`, and `U` values.

Vector Signal States

Input

The Virtuoso UltraSim simulator accepts the following signal states for input vector signals.

Table 13-3 Input Vector Signal States

Signal State	Description
`0`	Drive to ZERO (GND)
`1`	Drive to ONE (VDD)
`Z`, `z`	Floating to high-impedance
`X`, `x`	Drive to ZERO (GND)

Table 13-3 Input Vector Signal States, *continued*

Signal State	Description
`L`, `l`	Resistively drive to ZERO (GND)
`H`, `h`	Resistively drive to ONE (GND)
`U`, `u`	Drive to ZERO (GND)

The resistance values of `L` and `H` are set by the `hlz` statement, and the impedance value of `Z` is set by the `triz` statement.

Output

The Virtuoso UltraSim simulator accepts the following signal states for output vector signals.

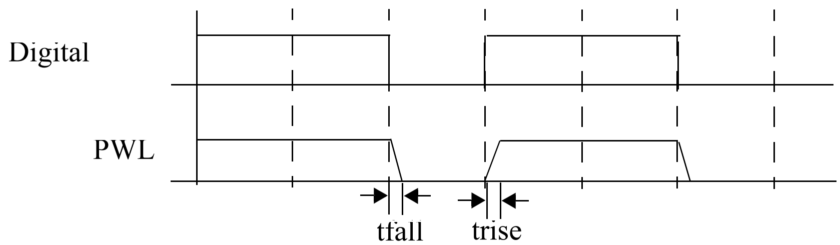
Table 13-4 Output Vector Signal States

Signal State	Description
`0`	Expects ZERO
`1`	Expects ONE
`Z`, `z`	Accepts any signal state
`X`, `x`	Accepts any signal state
`U`, `u`	Accepts any signal state

Digital Vector Waveform to Analog Waveform Conversion

The Virtuoso UltraSim simulator converts the digital vector waveform into a PWL waveform. The rising/falling edge occurs at the switching state point of the digital waveform, as shown in Figure 13-3 on page 655.

Figure 13-3 Conversion of Digital Waveform to PWL Waveform



Expected Output and Comparison Result Waveforms for Digital Vector Files

If a digital vector file contains output or bi-directional vectors, the Virtuoso UltraSim simulator generates two waveform files: One contains all the expected output vector waveforms as specified in the digital vector file and the other contains the waveforms from the comparison results.

You can use the following statement in the digital vector file to enable or disable the simulator from generating these waveforms (default is `1` or enabled).

```
.output_wf 0|1
```

The waveform format is defined by the `wf_format` option in the analog netlist file. A maximum of two waveform files are generated for one or more digital vector files. The expected waveform filename is `netlist.vecexp.trn` (PSF, FSDB, etc.) and the output vector is `signal_name_exp`. The comparison waveform filename is `netlist.vecerr.trn` (PSF, FSDB, etc.) and each comparison waveform is `signal_name_err`.

The comparison result values include,

0 – matched

1 – mismatched

X – ignored (output vector = X or bi-directional vector at input stage are possible causes)

In addition to the individual comparison result waveforms, the simulator generates a single `vec_error` waveform to indicate the overall comparison results. Waveform `vec_error` equals `1` when any of the individual comparison result waveforms also have a value of `1` (`X` is treated as `0`).

Example of a Digital Vector File

This is a basic digital vector file that shows how each Virtuoso UltraSim simulator statement is used.

```
; enable generation of expected output vectors and comparison result waveforms.
output_wf 1
; radix specifies the number of bit of the vector.
radix 2 2 4
; io defines the vector as an input or output vector.
io    i i o
; vname assigns the name to the vector.
vname A[1:0] B[1:0] P[3:0]
; tunit sets the time unit.
tunit ns
; trise specifies the rise time of each input vector.
trise 1
; tfall specifies the fall time of each input vector.
tfall 1
; vih specifies the logic high voltage of each input vector.
vih 2.5
; vil specifies the logic low voltage of each input vector
vil 0.0
; voh specifies the logic high voltage of each output vector
voh 2.0
; vol specifies the logic low voltage of each output vector
vol 0.5
0 0 0 x
200 3 3 x
400 1 2 0
600 2 1 9
800 3 1 2
1000  1  3   2
1200  2  2   3
1400  3  2   3
1600  2  3   4
1800  0  0   6
2000  0  0   7
```

Frequently Asked Questions

Can I replace the bidirectional signal with an input and output vector?

Bidirectional signals can be divided into two columns, one for an input vector and the other for an output vector (the enable signal is no longer needed). The same `vname` and signal name is used for the input and output vectors.

For the input stage, the value of the output vector must be `X` or `x` (output vector check is not performed). For the output stage, the value of the input vectors must be `Z` or `z` (no stimulus for this signal). For example:

```
radix 1 1 1 1
io i o i o
vname DI DO DQ DQ
tunit ns
0 0 1 0 x
100 1 0 1 x
200 0 1 0 x
300 0 0 z 1
400 1 1 z 0
500 0 0 z 1
```

How do I verify the input stimuli?

Use `.probe tran v(*) depth=1` to probe the top-level signals and then check the waveform outputs with the Virtuoso Visualization and Analysis or SimVision viewers.

Note: The signal names are case sensitive.

Review the log file to check if the signals defined in the digital vector file match those defined in the analog netlist file.

- When the signal is defined in the vector file, but not in the analog netlist file, a warning message appears stating that the VCD or VEC file is not defined in the netlist file.
- When the input signal is used in the analog netlist file, but does not match the one located in the vector file, check the list of dangling nodes or no DC path to ground in the log file.

How do I verify the vector check?

A `netlist.veclog` file is generated at the location specified by the Virtuoso UltraSim simulator `option-raw` statement if there are any vector checks. A `netlist.vecerr` file is also generated when errors occur during the vector check. Refer to these two files for detailed information about the vector check.

When the signal is defined in the vector file, but not in the analog netlist file, the simulator issues a warning message in the log file that states the signal node is missing from the netlist file.

In addition, the simulator generates two waveform files: One contains all the expected output vector waveforms as specified in the digital vector file and the other contains the waveforms from the comparison results.

14

Verilog Value Change Dump Stimuli

This chapter introduces the Verilog® value change dump (VCD) and extended VCD (EVCD) file formats, as used in the Virtuoso® UltraSim™ simulator, and provides illustrations to explain the signal information file.

The VCD file (ASCII format) contains information about value changes for selected variables in the circuit design. The Virtuoso UltraSim simulator supports two types of VCD files:

- **Four-state** – represents variable changes in `0`, `1`, `x` (unknown or "not needed") and `z` (tri-state) without providing strength information and port direction
- **Extended** – represents variable changes in all states and provides strength information and port direction

  For more information about EVCD file format, refer to "Enhanced VCD Commands" on page 703.

Processing the Value Change Dump File

To process VCD or EVCD files in the Virtuoso UltraSim simulator, the following command cards need to be specified in the netlist file:

Spectre Syntax

```
vcd_include "vcd_filename" "signal_info_filename" [autostop=true|false]
     [insensitive=yes|no]
```

SPICE Syntax

```
.vcd "vcd_filename" "signal_info_filename" [autostop=true|false]
     [insensitive=yes|no]
```

or

Spectre Syntax

```
evcd_include "evcd_filename" "signal_info_filename" [autostop=true|false]
     [insensitive=yes|no]
```

SPICE Syntax

```
.evcd "evcd_filename" "signal_info_filename" [autostop=true|false]
     [insensitive=yes|no]
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.vcd` or `.evcd`).

The Virtuoso UltraSim simulator replaces the end time in the `.tran` or `tran` statement with the time specified in the `.vcd`/`.evcd` file when `autostop=true` (default is `false`). If `false` is selected, the simulation time specified in the `.tran` or `tran` statement remains unchanged. For more information on `autostop`, refer to Chapter 13, "Digital Vector File Format".

Each `vcd` or `evcd` card can only specify one VCD or EVCD file. If a netlist file needs to include multiple VCD or EVCD files, multiple `vcd` or `evcd` cards must be used. For example, if a netlist file contains three VCD files, it needs three `vcd` cards (use the same netlist file format for EVCD files).

Spectre Syntax:

```
Card 1: vcd_include "file1.vcd" "file1.signal"
Card 2: vcd_include "file2.vcd" "file2.signal"
Card 3: vcd_include "file3.vcd" "file3.signal"
```

SPICE Syntax:

```
Card 1: .vcd "file1.vcd" "file1.signal"
Card 2: .vcd "file2.vcd" "file2.signal"
Card 3: .vcd "file3.vcd" "file3.signal"
```

Note: A netlist file can include multiple VEC, VCD, and ECVD files.

VCD and EVCD formats are widely used in digital circuit design and contain different kinds of information for transistor level simulation.You need to provide signal information, such as timing characteristics, voltage threshold, and driving ability of input signals for each VCD or EVCD file. Since VCD and EVCD formats are compatible, the Virtuoso UltraSim simulator can share the same signal information file.

The Virtuoso UltraSim simulator handles the VCD and EVCD file content as case insensitive, except when called in Virtuoso Spectre® mode. For Spectre mode, use the `-spectre` option or input file name extension `*.scs`. Additional case sensitivity can be set using the `insensitive` option.

VCD Commands

VCD File Format

Continuous Line

The continuous line symbol is the forward slash (`\`) sign and is rarely used in the VCD file. The beginning of a card is indicated by the `$command` keyword (for example, `.$var`), and ends with the `$end` keyword. If an identifier is longer than 1024 characters, and does not fit into a single line, the `\` sign must be used to continue the line.

$comment

`$comment` *`comments`* `$end`

Description

Comments need to be enclosed within the `$comment` and `$end` commands.

Example

```
$comment
This is a test line.

$end
```

Definition Commands

In the definition section, the VCD command keywords used to indicate the start of a card begin with a dollar (`$`) sign. VCD definition commands include: `$comment`, `$date`, `$enddefinitions`, `$scope`, `$timescale`, `$upscope`, `$var`, and `$version`. The end of a card is marked with an `$end` keyword. Multiple lines can be placed between the `$command` and `$end` commands.

The Virtuoso UltraSim simulator supports the following VCD definition section commands:

- $date on page 663
- $enddefinitions on page 664
- $scope on page 665
- $timescale on page 666
- $upscope on page 667
- $var on page 668
- $version on page 670

$date

`$date` *`date`* `$end`

Description

`The $date` command is used to specify the date of the VCD file created. The Virtuoso UltraSim simulator accepts this command card, but does not process it.

Example

```
$date May 7, 2001 $end
```

$enddefinitions

```
$enddefinitions $end
```

Description

The `$enddefinitions` command indicates where the definition section of the VCD file ends. This command card tells the Virtuoso UltraSim simulator to treat the rest of the VCD file as the data section. If this command card is missing, the Virtuoso UltraSim simulator parses the data section incorrectly and issues an error message.

Example

```
$enddefinitions $end
```

$scope

`$scope` *`[scope_type] [scope_name]`* `$end`

Description

The `$scope` command card switches from the current circuit level to a lower circuit level in the design hierarchy.

Example

```
$scope module module1 $end
```

It is important to note:

- The Virtuoso UltraSim simulator ignores `scope_type`, but the command must still be specified to maintain consistency with standard VCD format.
- A matching `$upscope` card must be specified in the VCD file definition section to switch the scope back to the current scope.

$timescale

`$timescale` *`[number] [time_unit_prefix]`* `$end`

Description

The `$timescale` command is used to specify the time scale. This time scale applies to all time values in the VCD file, and to its signal information file. The default time is 1 ns.

Arguments

number	A positive double number
time_unit_prefix	The unit prefix specified in "Unit Prefix Symbols" on page 62.

Example

```
$timescale 1 ns $end
```

or

```
$timescale 1ns $end
```

produces the same result, telling the Virtuoso UltraSim simulator to set the time scale to 1 ns.

$upscope

`$upscope $end`

Description

The `$upscope` command card switches from the current circuit level to an upper circuit level in the design hierarchy.

Example

```
$upscope $end
```

Note: This card must be used after the `$scope` card to switch the scope back to the top scope.

$var

`$var` *`[var_type] [size] [identifier] [reference]`* `$end`

or

`$var` *`[var_type] [size] [identifier] [reference] [index_range]`* `$end`

Description

The `$var` command defines the bus to be dumped into the data section.

Arguments

var_type	The bus type (the Virtuoso UltraSim simulator ignores this information)
size	Number of bits in this bus specified as a decimal number
identifier	The identifier used in the data section; it can be `a` or a combination of printable ASCII characters
reference	The name of the bus
index_range	The index range of the bus in the following formats: ■ [MSI:LSI] ■ [LSI:MSI] ■ [index] if the size is 1 **Note:** MSI, LSI, and index must be an integer

Note: If the size is larger than 1, and the *index_range* is not specified, the Virtuoso UltraSim simulator assigns an `index_range` of [size-1:0].

The name of the bit is a combination of the reference and the index.

Examples

In the following example

```
$var reg 4 % regA [0:3] $end
```

the names of the four bits in bus `regA` are `regA[0]`, `regA[1]`, `regA[2]`, and `regA[3]`. The netlist file referencing these VCD sources must match these names.

In the next example

```
$var reg 1 & b [0] $end
```

the name of the bit is `b[0]`. The card defined a bit, not a bus, as a size of `1`.

In the next example

```
$var reg 1 & c $end
```

the name of the bit is `c`.

In the next example

```
$var reg 4 % regA $end
```

the names of the four bits in bus `regA` are `regA[3]`, `regA[2]`, `regA[1]`, and `regA[0]`. The netlist file referencing these VCD sources must match the names.

$version

```
$version version $end
```

Description

The `$version` command is used to specify the version of the VCD file created. The Virtuoso UltraSim simulator accepts this command card, but does not process it.

Example

```
$version UltraSim B2001.2.10 $end
```

Data Commands

In the VCD data section, a time point that starts with a number (`#`) sign (for example, #100) indicates the beginning of a new card. This card continues until it reads the line before the next card (cards can have multiple lines). It can also contain data values and a command block. A command block begins with one of the following command keywords, `$comment` or `$time_value`, and ends with `$end`.

data

Description

Data is divided into two types that each have their own format:

- Bus data
- Bit data

The bus data is for buses defined by $var, with a size greater than one. The bit data applies to the bus defined by $var, with a size equal to one. The bus data format is *`Bvalue bus_identifier`*.

Examples

In the following example

```
b0101 %
```

the bus with identifier `%` (defined by `$var`) has a binary value of 0101. The bit data format is *`value bus_identifier`*.

In the next example

```
1&
```

the one bit bus with identifier `&` (defined by `$var`) has a binary value of 1.

The valid characters for specifying the value are: 0, 1, x, X, z, and Z. Of these characters, x and X are treated as 0 for the input signal, and do not need a vector check for the output signal. The z and Z characters are treated as floating to high-impedance for the input signal, and do not need a vector check for the output signal.

time_value

`#time_value`

Description

Each time value (point) is the beginning of a card in the data section.

Example

```
#100
```

Time value is equal to 100 time units (time unit is defined by $timescale).

Signal Information File

Note: The information in this section is applicable to both VCD and EVCD formats.

The signal directions are specified in the signal information file. To input signals, the Virtuoso UltraSim simulator applies stimuli to the simulation. The values of the output signals are used to perform a vector check against the simulation results, and vector errors are generated if mismatches occur. The enable signals are required to control the bi-directional signal behavior as input or output signals. All other signals in the VCD or EVCD file, not specified in the signal information file, are ignored by the simulator (warning messages are issued for these signals, based on the scopes specified in the signal information file).

The time scale of the time related cards in the signal information file are controlled by the `$timescale` card in the VCD file. For example, if `$timescale` is set to 1 ns and `.tdelay` to 2, a delay of (2 * 1ns) occurs.

The signal name in the signal information file can be specified by using the bus name or a wildcard (for more information about wildcards, see "Wildcard Rules" on page 55). The signal name specified in the VCD file is in bus/bit format. Table 14-1 on page 673 provides examples.

Table 14-1 Signal Name in VCD File Corresponds to Signal Information File

VCD File	Signal Information File
P[0] ... P[15]	P[0:15]
P[0] ... P[15]	P[*]
Q[63:32] Q[31:0]	Q[63:0]

The example in Table 14-2 on page 673 shows an incorrect naming format.

Table 14-2 Incorrect Naming Format in Signal Information File

VCD File	Signal Information File
A[0:15]	A[0:7] A[8:15]

Note: By default, the Virtuoso UltraSim simulator creates flat mapping between the VCD and analog netlist files (the `.hier` statement can be used to switch to hierarchical name mapping to precisely match signals).

Signal Information File Format

Comment Line

A comment line begins with asterisk (`*`) or dollar (`$`) signs.

Continuous Line

A continuous line is indicated by a plus (`+`) sign.

The maximum length of a line is 1024 characters. If a card is longer than 1024 characters, you need to use the continuous line for the card.

To process VCD and EVCD formats, the following signal characteristics need to be defined in the signal information file:

- Signal Matches on page 675
- Signal Timing on page 687
- Voltage Threshold on page 693
- Driving Ability on page 698

Signal Matches

In this section, the following statements define the signal matches between the VCD and analog netlist files, signal directions, and output vector check.

.alias

`.alias` *`target_busname alias_name`*

Description

The `.alias` statement is used to modify the name of the signal bus in the VCD/EVCD file to match the signal name in the netlist file.

Note: For more information about using the `.alias` statement in hierarchical signal name mapping, refer to "Hierarchical Signal Name Mapping" on page 699.

Examples

The following example

```
.alias *[*] *<*>
```

tells the Virtuoso UltraSim simulator to change the square brackets to angle brackets (see Table 14-3 on page 676).

Table 14-3 .alias *[*] <*> Names

Bus Name	Signal Name
a[0:3]	a<0>, a<1>, a<2>, a<3>
vec[3:0]	vec<3>, vec<2>, vec<1>, vec<0>

In the next example

```
.alias sig_*[*] vec_*<*>
```

tells the simulator to change the square brackets to angle brackets (see Table 14-4 on page 676).

Table 14-4 .alias sig_*[*] vec_*<*> Names

Bus Name	Signal Name
sig_1[0:3]	vec_1<0>, vec_1<1>, vec_1<2>, vec_1<3>
sig_bus1[3:0]	vec_bus1<3>, vec_bus1<2>, ve_bus1c<1>, vec_bus1<0>

In the next example

```
.alias sig_* vec_*
```

tells the simulator to change the prefix of the signal names from `sig_` to `vec_` (see Table 14-5 on page 677).

Table 14-5 .alias sig_* vec_ * Names

Bus Name	Signal Name
`sig_1`	`vec_1`
`sig_2`	`vec_2`

.scope

`.scope` *`scope_name1 [scope_name2 ... scope_nameN]`*

Description

The `.scope` statement specifies the target scope located in the definition section of the VCD file. Only the signals defined in the specified scope are processed. Multiple `.scope` statements in a signal information file are supported.

Arguments

`scope_name`	The name of the scope specified in the VCD file by the `$scope` card.

Note: Each scope name causes the Virtuoso UltraSim simulator to process the signals located in the specified scope, but not the signals located in its parent or child scopes.

Example

```
.scope module1 module2
```

tells the Virtuoso UltraSim simulator to process the signal located in scope `module1` and `module2`.

.in

```
.in signal_name1 signal_name2 ... signal_nameN
```

Description

The `.in` statement defines the specified bus as the input bus.

Arguments

name	The name of the bus specified in the VCD file

Example

VCD file:

```
$var reg 4 % b [0:3] $end
$var reg 1 * a $end
$var reg 1 & c [4] $end
$var reg 4 % d [0:3] $end
```

Signal information file:

```
.in b[0:3] a c[4] d[*]
```

Defines `b[0:3]`, `a`, `c[4]`, and `d[0:3]` as the input signals.

.out

```
.out signal_name1 signal_name2 ... signal_nameN
```

Description

The `.out` statement defines the specified bus as the output bus.

Arguments

`name`	The name of the bus specified in the VCD file

Example

In the VCD File:

```
$var reg 4 % b [0:3] $end
$var reg 1 * a $end
$var reg 1 & c [4] $end
$var reg 4 % d [0:3] $end
```

In the signal information file

```
.out b[0:3] a c[4] d[*]
```

Defines `b[0:3]`, `a`, `c[4]`, and `d[0:3]` as the output signal.

.bi

`.bi` *`'enable_signal_expr' signal_name1 signal_name2`* `...` *`signal_nameN`*

Description

The `.bi` statement defines the specified bus as the bidirectional bus.

The enable signal can be from a VCD/EVCD or an analog netlist file. When an enable signal is from an analog netlist file, it can also be defined as an output signal for a vector check or only used as an enable signal. If a VCD signal is used as an enable signal, it must be declared an input using the .in statement and located in the VCD file. Different from enable statements in the vector file, the logic voltage threshold of an analog enable signal is defined by the .voh and .vol statements.

Note: The enable signal cannot be defined as a bidirectional signal.

The .alias statement can be used to perform name mapping for the enable signal. In hierarchical signal name mapping (`.hier 1`), a hierarchical structure for the analog netlist file is supported for the enable signal. A period (`.`) can be used as the hierarchical delimiter to specify the hierarchical signal, and the hierarchical delimiter can be mapped to other delimiters by the `.alias` statement.

Bit-wise logic operators are supported in an enable signal expression: `&` (AND), `|` (OR), `^` (XOR), and `~` (NOT). Additional operators can be created using a combination of the supported operators. The order of processing for the logic operators is NOT > AND > OR, XOR (OR and XOR are processed at the same time). You can use parentheses `()` around the operators to change the processing order.

Arguments

`'enable_signal_expr'`	The expression for the enable signals. The bidirectional bus switches to output mode only when its value is high. **Note:** You need to use single quotation marks `''` for enable signal expressions.
`signal_name`	The name of the bus specified in the VCD/EVCD file.

Examples

The following example

VCD file:

```
$var reg 4 % b [0:3] $end
$var reg 1 & en $end
```

Signal information file:

```
.bi en b[0:3]
```

defines `b[0:3]` as the bidirectional signal, which is controlled by the `en` signal. When `en` is high, `b[0:3]` becomes the output signal.

In the next example

VCD file:

```
$var reg 4 * myBi [0:3] $end
$var reg 1 # en $end
```

Signal information file:

```
bi ~en myBi[0:3]
```

defines `myBi[0:3]` as the bidirectional signal, which is controlled by the `en` signal. The enable signal is appended with a tilde (~) sign, so unlike the first example, the `en` is now high, `myBi[0:3]` becomes the input signal, and vice versa.

In the next example

VCD file:

```
$var reg 4 * myBi_1 [0:3] $end
$var reg 1 # en $end
```

Signal information file:

```
.hier 1
.bi 'en & (ana_en1 ^ X1.ana_en2)' myBi_1[0:3]
.voh 1.5 ana_en* X1.ana_en*
.vol 0.8 ana_en* X1.ana_en*
```

defines `myBi_1[0:3]` as the bidirectional signal, which is controlled by the expression `'en & (ana_en1 ^ X1.ana_en2)'`. When the value of the expression is high, `myBi_1[0:3]` becomes the input signal, and vice versa. The `.voh` and `.vol` statements define the logic voltage threshold of the two analog enable signals.

.chk_ignore

`.chk_ignore start_time end_time [signal_name1 ... signal_nameN]`

Description

The `.chk_ignore` statement specifies a window used to ignore output vector checks for a VCD file. You can provide the signal names in order to apply this statement locally. In addition to hierarchical mapping, the hierarchical structure is also given in the signal names. The `start_time` and `end_time` arguments must be specified. To define multiple time windows for ignoring output vector checks, use multiple `chk_ignore` statements.

Arguments

`start_time`	Defines the start time for the window used to ignore the output vector checks (use the $timescale card in the VCD file to set the time scale).
`end_time`	Defines the end time for the window used to ignore the output vector checks. You can use `end_time=-1` to ignore the entire transient time (use the $timescale card in the VCD file to set the time scale).

Example

VCD file:

```
$timescale 1ns $end
```

Signal information file:

```
.hier 1
.chk_ignore 0 100 X1.out1 Top.digital.pout[*]
.chk_ignore 300 500 X1.out1 Top.digital.pout[*]
.chk_ignore 0 -1 out[*]
```

tells the Virtuoso UltraSim simulator to ignore the output vector check for signals `X1.out1` and `Top.digital.pout[*]` in the time windows 0 ns to 100 ns and 300 ns to 500 ns, and to ignore the entire transient time for the signal `out[*]`.

.chkwindow

```
.chkwindow start_time end_time steady [period=const [first=const] ] [ signal_name1
     ... signal_nameN ]
```

Description

The `.chk_window` statement specifies a window for output vector checking. The Virtuoso UltraSim simulator only checks the signal states within this window. The signal states outside the window are ignored. The checks occur at every time point specified in the VCD/EVCD file or as defined by the `period` and `first` arguments.

Setting the `period` argument activates periodic window checking. If `period` is not defined, the `first` argument is ignored by the simulator.

Note: To activate periodic window checking, you need to include the `"period="` and `"first="` keywords.

Arguments

`start_time`	Defines the window start time at which the window starts at time `vec_time-start_time`. If the `period` argument is defined, `vec_time` is the first time point defined by the `first` argument, and the vector checks are repeated according to the value of `period`. If the `period` argument is not defined, `vec_time` is the time point defined in the VCD/EVCD file.
`end_time`	Defines the window end time at which the window ends at time `vec_time+end_time`.
`steady = 0 | 1`	If set to `0`, then the vector check passes as long as the signal has reached the desired state once. If set to `1`, then the signal remains in the desired state for the entire window period to pass the vector check.
`period`	Activates periodic window checking and defines its time period.
`first`	Defines the first check point for periodic window checking (only valid when the `period` argument is also defined).

Examples

In the following example

VCD file:

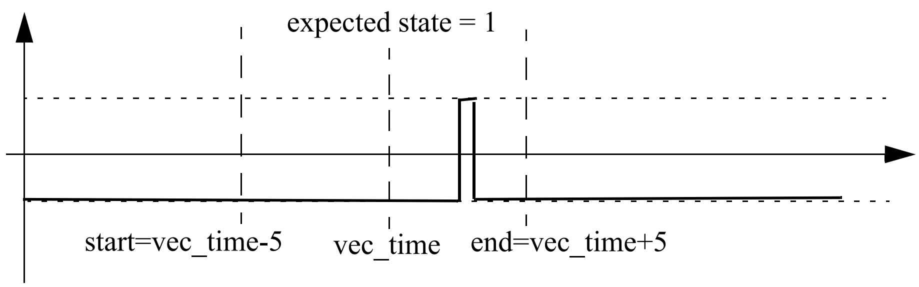
```
$timescale 1ns $end
```

Signal information file:

```
.chkwindow 5 5 0
```

The `.chkwindow` statement is set to `0`, so the waveform passes the vector check (see Figure 14-1 on page 685).

Figure 14-1 Vector Check with `.chkwindow` Set to 0



In the next example

VCD file:

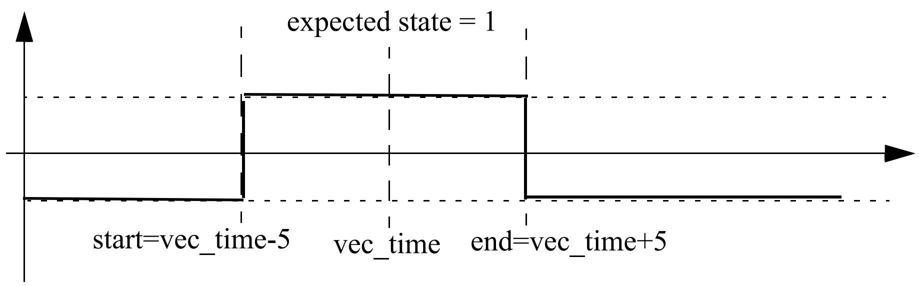
```
$timescale 1ns $end
```

Signal information file:

```
.chkwindow 5 5 1
```

The `.chkwindow` is statement set to `1` and the signal remains at that state for the entire window period, in order to pass the vector check (see Figure 14-2 on page 685). If the signal matches Figure 14-1 on page 685, the vector check fails.

Figure 14-2 Vector Check with `.chkwindow` Set to 1



In the next example

VCD file:

```
$timescale 100ps $end
```

Signal information file:

```
.chkwindow 5 5 1 period=100 first=20 p[*] out_*
```

tells the simulator to activate periodic window checking for signals `p[*]` and `out_*`. The vector check points start at 2 ns and repeats every 10 ns.

In the next example

```
.hier 1
.chkwindow 5 5 1 first=20 X1.Xana.p[*] X1.Xdigital.Xcore.out_*
```

tells the simulator to ignore the `first` argument because a valid `period` argument has not been specified. The other arguments are used in the simulation.

Signal Timing

The signal timing characteristics (delay, rise and fall time) are defined in this section.

- .idelay on page 688
- .odelay on page 689
- .tdelay on page 690
- .tfall on page 691
- .trise on page 692

Note: The Virtuoso UltraSim simulator checks whether the values of the `.trise` and `.tfall`. statements are reasonable (warning message is issued when the defined value is too small or large).

.idelay

```
.idelay time_value [signal_name1 ... signal_nameN]
```

Description

The `.idelay` statement specifies the delay time for the corresponding input signals. If a bidirectional signal is specified, this only applies to the input stage of the bidirectional signal. The default value is 0.0, if `.idelay` and `.tdelay` are not set.

Example

In the following example

VCD file:

```
$timescale 1ns $end
```

Signal information file:

```
.idelay 5
```

All input signals have a time delay of 5 ns.

In the next example

VCD file:

```
$timescale 100 ps $end
```

Signal information file:

```
.hier 1
.scope Xtop.XI1
.in a[0:3] b
.trise 0.1 Xtop.XI1.a* Xtop.XI1.b
```

The `Xtop.XI1.a[0:3]` and `Xtop.XI1.b` input signals have a rise time of 10 ps.

.odelay

```
.odelay time_value [signal_name1 ... signal_nameN]
```

Description

The `.odelay` statement specifies the delay time for the corresponding output signals. If a bidirectional signal is specified, this only applies to the output stage of the bidirectional signal. The default value is 0.0, if `.odelay` and `.tdelay` are not set.

Example

VCD file:

```
$timescale 1ns $end
```

Signal information file:

```
.odelay 5
```

All output signals have a time delay of 5 ns.

.tdelay

```
.tdelay time_value [signal_name1 ... signal_nameN]
```

Description

The `.tdelay` statement specifies the delay time for the corresponding input, output, and bidirectional signals. The default value is 0.0, if `.tdelay`, `.idelay`, and `.odelay` are not specified.

Example

VCD file:

```
$timescale 1ns $end
```

Signal information file:

```
.tdelay 5
```

All signals have a time delay of 5 ns.

.tfall

```
.tfall time_value [signal_name1 ... signal_nameN ]
```

Description

The `.tfall` statement specifies the fall time of the input signal. If `.tfall` is not specified, the default value is 0.1 n.

Examples

In the following example

VCD file:

```
$timescale 1 ns $end
```

Signal information file:

```
.tfall 0.1
```

all input signals have a fall time of 0.1 ns.

In the next example

VCD file:

```
$timescale 1 ns $end
```

Signal information file:

```
.in a[0:3] b
.tfall 0.1 a[0:3] b
```

input signals `a[0:3]` and `b` have a fall time of 0.1 ns.

.trise

```
.trise time_value [signal_name1 ... signal_nameN]
```

Description

The `.trise` statement specifies the rise time of the input signal. If `.trise` is not specified, the default value is 0.1 n.

Examples

In the following example

VCD file:

```
$timescale 1 ns $end
```

Signal information file:

```
.trise 0.1
```

all input signals have a rise time of 0.1 ns.

In the next example

VCD file:

```
$timescale 1 ns $end
```

Signal information file:

```
.in a[0:3] b
.trise 0.1 a[0:3] b
```

input signals `a[0:3]` and `b` have a rise time of 0.1 ns.

Voltage Threshold

As digital vector format, the voltage threshold for logic low and high can be specified using the following statements to convert the input vectors to stimuli or to perform an output vector check.

- .vih on page 694
- .vil on page 695
- .voh on page 696
- .vol on page 697

.vih

```
.vih voltage_value [signal_name1 ... signal_nameN]
```

Description

The `.vih` statement specifies the logic high voltage of the input signal. If `.vih` is not specified, the default voltage is 3.3.

Examples

In the following example

```
.vih 5.0
```

tells the Virtuoso UltraSim simulator all input signals have a logic high voltage of 5.0.

In the next example

```
.in a[0:3] b
.vih 5.0 a[*] b
```

tells the simulator input signals `a[0:3]` and `b` have a logic high voltage of 5.0.

.vil

```
.vil voltage_value [signal_name1 ... signal_nameN]
```

Description

The `.vil` statement specifies the logic low voltage of the input signal. If `.vil` is not specified, the default voltage is 0.0.

Examples

In the following example

```
.vil 1.0
```

tells the Virtuoso UltraSim simulator all input signals have a logic low voltage of 1.0.

In the next example

```
.in a[0:3] b
.vil 1.0 a[0:3] b
```

tells the simulator input signals `a[0:3]` and `b` have a logic low voltage of 1.0.

.voh

```
.voh voltage_value [ signal_name1 ... signal_nameN ]
```

Description

The `.voh` statement specifies the logic high voltage of signals defined as outputs and signals from the analog netlist file that are used in the signal information file. If `.voh` is not specified, the default voltage is 3.3.

Examples

In the following example

```
.voh 5.0
```

tells the Virtuoso UltraSim simulator all output signals have a logic high voltage of 5.0.

In the next example

```
.out out[0:3] outA
.voh 5.0 out[*] outA
```

tells the simulator output signals `out[0:3]` and `outA` have a logic high voltage of 5.0.

.vol

```
.vol voltage_value [signal_name1 ... signal_nameN]
```

Description

The `.vol` statement specifies the logic low voltage of signals defined as outputs and signals from the analog netlist file that are used in the signal information file. If `.vol` is not specified, the default voltage is 0.0.

Examples

In the following example

```
.vol 1.0
```

tells the Virtuoso UltraSim simulator all output signals have a logic low voltage of 1.0.

In the next example

```
.out out[0:3] outA
.vol 1.0 out[0:3] outA
```

tells the simulator output signals `out[*]` and `outA` have a logic low voltage of 1.0.

Driving Ability

For input stimuli, the output resistance of VCD/EVCD sources can affect Virtuoso UltraSim simulation results. To specify the driving ability of VCD/EVCD sources, use the following statements:

.outz

`.outz` *`resistance [signal_name1 ... signal_nameN]`*

Description

The `.outz` statement specifies the output resistance for corresponding input signals. If `.outz` is not specified, the default value is 0.01.

Example

```
.outz 1meg
```

All input signals have an output resistance of 1 Megaohm.

.triz

`.triz` *`resistance [signal_name1 ... signal_nameN]`*

Description

The `.triz` statement specifies the output impedance when the corresponding input signals are in tri-state. If `.triz` is not specified, the default value is 1,000 Meg.

Example

```
.triz 500meg
```

All input signals have an output impedance of 500 Megaohms.

Hierarchical Signal Name Mapping

Hierarchical signal name mapping can be used to precisely match signals between the VCD and analog netlist files, and is defined by the Virtuoso UltraSim simulator hier statement:

```
.hier 0 | 1
```

Description

The `.hier` statement is used to specify hierarchical names in both the VCD and analog netlist files.

Arguments

`hier`	If `hier=0`, the Virtuoso UltraSim simulator creates flat mapping between the VCD and analog netlist files (default). To maintain backward compatibility, the hierarchical delimiter is regarded as part of the signal name. If `hier=1`, the simulator applies the hierarchical names to the VCD and analog netlist files. The VCD file stimuli are no longer limited to the top level of the analog netlist file. In the VCD info file, the complete hierarchical structure needs to be added to the `.scope` statement (the hierarchical signals in the analog netlist file are mapped according to the information provided by the `.alias` statement).

The key differences between flat and hierarchical mapping include:

- To match hierarchical signals in the VCD file, the complete hierarchical structure needs to be specified in the `.scope`, `.alias`, and `.chkwindow` statements, as well as in the signal characteristic statements. For flat mapping, only the signal names are needed.
- For hierarchical signal name mapping, the statements to define the port direction of the signal are related to the `.scope` statement, which includes the `.in`, `.out`, and `.bi` statements. When using flat mapping, the multiple scopes defined by the `.scope` statement are regarded as set. The Virtuoso UltraSim simulator searches all of the scopes to perform a signal match and outputs an error message when the same signal name is defined in more than one VCD scope.
- The hierarchical structure of the analog netlist file is specified by the `.alias` statement when performing hierarchical mapping.

Enhanced Statements

.scope and .in/.out/.bi

When using the Virtuoso UltraSim simulator to apply hierarchical names to the VCD/EVCD and analog netlist files (`.hier 1`), the hierarchical structure in the VCD/EVCD file must be clearly defined with the `.scope` statement. The `.in`, `.out`, and `.bi` statements are used to define the signal name in the specified `.scope` statement and cannot contain the hierarchical structure. For the `.bi` statement, the enable signal needs to contain the hierarchical path because the signal may belong to a different scope.

The VCD/EVCD signal info file supports multiple `.scope` statements. The effective scope of each `.scope` statement is affected by the other statements, requiring the `.in`, `.out`, and `.bi` statements to be in the correct location.

Examples

In the VCD file:

```
$scope module top $end
$scope module digital $end
$var reg       1 !    din_1  $end
$var reg       1 "    en  $end
$scope module drv $end
$var reg       1 !    mid[1]  $end
$var reg       1 "    inout  $end
$upscope $end
$upscope $end
$upscope $end
```

In the VCD info file:

```
.hier 1
.scope top.digital
* The effective scope is top.digital
.in din_1 en
* The scope top.digital is ended by the next .scope statement
.scope top.digital.drv
* The effective scope is changed to top.digital.drv
.out mid[*]
.bi top.digital.en inout
```

.alias

```
.alias targ_signame alias_signame
```

The hierarchical structures that are defined in the `.scope` statement are used only for the VCD/EVCD file, so the `.alias` statement is needed to map the signals to the lower circuit levels of the analog netlist file.

Note: If multiple `.alias` statements define the mapping relationship for a signal, the last `.alias` statement is used by the simulator, and the other statements are overwritten.

Arguments

`targ_signame`	Specifies the hierarchical signal names (VCD/EVCD format) that are already defined in the .scope statement and `.in`/`.out`/`.bi` statements. The hierarchical delimiter is represented by a period (`.`).
`alias_signame`	Specifies the hierarchical signal names of the analog netlist file. The `hier_delimiter` option defines the hierarchical delimiter.

Examples

In the following example

```
.alias TOP.module1.in1 X1.in1
.scope Top.module1
.in in1
```

tells the Virtuoso UltraSim simulator to map the `TOP.module1.in1` defined in the `.scope` and `.in`/`.out`/`.bi` statements to the signal `in1` of instance `X1` in the analog netlist file.

In the next example

```
.alias *[*] *<*>
.alias Top.module1.sig_*[*] X1.vec_*<*>
.scope Top.module1
.out sig_[0:15]
.scope digital_block
.in datain[*]
```

tells the simulator for the `Top.module1.sig_[0:15]` signals, the second `.alias` statement overwrites the first `.alias` statement and maps the signals to `X1.vec_<0>, ... X1.vec_<15>`. Since only the first `.alias` statement matches the `digital_block.datain[*]` signals, they are mapped to `digital_block.datain<*>` of the analog netlist file.

In the next example

- Analog netlist file:

  ```
  .usim_opt hier_limiter = %
  ```

- VCD/EVCD info file:

  ```
  .alias TOP.module1.sig_[*] X1%Xdrv%vec_<*>
  .scope Top.module1
  .out sig_*
  ```

Note: The hierarchical delimiter for the analog netlist file is percent (`%`), not period (`.`).

Hierarchical Signal Names

For hierarchical mapping, the signal names in the `.alias` and `.chkwindow` statements, as well as in the signal characteristic statements, must have the hierarchical structure in the VCD file.

Examples

```
.hier 1
.chkwindow -1 5 1 Top.X1.out*
.vih 1.2 X1.in[*]
.vol 0.2 X1.dout1 Xdig.X2.dout1
.tfall 0.2 Xana.X1.in* Xdig.din*
.outz 100 X1.in[*]
```

Enhanced VCD Commands

Since the EVCD and VCD formats are similar, only the key EVCD format differences will be discussed in this section:

- Signal Strength Levels on page 703
- Value Change Data Syntax on page 703
- Port Direction and Value Mapping on page 705

Signal Strength Levels

Verilog HDL allows scalar net signal values to have a full range of unknown values, and different levels of strength or combinations of levels of strength. For logic operation, multiple-level logic strength modeling resolves combinations of signals into known or unknown values, allowing the behavior of hardware to be represented with improved accuracy.

EVCD signal strength can be defined by eight values, ranging from `0` (weakest) to `7` (strongest). The most commonly used values are `0` and `6`. For example, for logic value 0

```
<0_strength_component> =6 , <1_strength_component> =0
```

and for logic value 1

```
<0_strength_component> =0 , <1_strength_component> =6
```

Note: Logic strength levels are not defined in VCD files because only four states are supported.

The Virtuoso UltraSim simulator ignores strength information (minimal impact on most CMOS circuit designs). If you want to preserve driver strength during simulation, specify .outz in the signal information file for specific signals with different output resistances.

For more information about logic strength modeling, refer to *IEEE Std 1364-2001*.

Value Change Data Syntax

The EVCD `data` command is different from the one used with VCD because the EVCD version can provide strength information and additional signal states.

data

```
p[port_value] [0_strength_component] [1_strength_component] [identifier_code]
```

Description

The value change section shows the actual value changes at each simulation time increment. Only variables that change value during a time increment are listed. In the EVCD file, strength information and a larger number of value states with port direction are presented in the value change section. The arguments for the EVCD `data` command are listed below.

Arguments

p	Key character that indicates a port. **Note:** There is no space between `p` and `port_value`.
port_value	State character which contains information about driving direction and the value of the port. The state characters are described in "Port Direction and Value Mapping" on page 705 (see tables).
0_strength_component	One of the eight Verilog strength values indicating the `strength0` component of the value (the Virtuoso UltraSim simulator ignores this value).
1_strength_component	One of the eight Verilog strength values indicating the `strength1` component of the value (the Virtuoso UltraSim simulator ignores this value).
identifier_code	The identifier code for the port, which is defined in the $var construct for the port.

Examples

The following example

```
pU 0 7 &lt;0
```

tells the Virtuoso UltraSim simulator the one bit bus with identifier `<0` (defined by `$var`) has a binary value of `U`, and the strength of 0 component is `0` and the strength of 1 component is `7`.

In the next example

```
pCCC 667 667 !
```

tells the simulator the bus with identifier `!` (defined by `$var`) has a binary value of `CCC`, and the strength of 0 component is `667` and the strength of 1 component is `667`. There is more than one driver on this port and the resolved value is `CCC`.

Port Direction and Value Mapping

The port value in the EVCD file contains port direction information, which helps the Virtuoso UltraSim simulator distinguish some of the `x` states, apply stimuli for input signals, or perform a vector check for output signals.

Note: To generate the EVCD file, the port directions of the circuit simulated by the Virtuoso UltraSim simulator need to be consistent with the port directions of the device under test (DUT).

input Direction

Given an DUT and a test fixture, the driving direction is `input` if the text fixture drivers are driving a non-tristate value and the drivers inside the DUT are tri-state. The resolved value is mapped in Table 14-6.

When reading the mapping information in the following tables, it is important to note:

- Declared `in` and declared `out` indicates the signal is defined as `input` and `output` in the signal information file. The term `active` implies the drivers are in a non-tristate condition.
- Because the conflicting states of the signal value are converted to `x` in the VCD file, they are regarded as “not needed” and the Virtuoso UltraSim simulator does not perform a vector check when the signals are specified as `output`.
- Combining the port direction for the signal value in the EVCD file and the specified direction in the signal information file, the Virtuoso UltraSim simulator can distinguish the `input` and `output` values of a signal and perform a vector check when it is specified as `output` in the signal information file.

Table 14-6 input Value Mapping

Port Value	Declared `in`	Declared `out`	Declared `bi` Enable = 0	Declared `bi` Enable = 1	Mapped VCD Value
`D`	`0` input	No check	`0` input	No check	Low – only one active driver to the port
`d`	`0` input	No check	`0` input	No check	Low – two or more active drivers to the port (may be conflicts, yet resolved value is low)

Table 14-6 input Value Mapping, *continued*

Port Value	Declared `in`	Declared `out`	Declared `bi` Enable = 0	Declared `bi` Enable = 1	Mapped VCD Value
`U`	`1` input	No check	`1` input	No check	High – only one active driver to the port
`u`	`1` input	No check	`1` input	No check	High – two or more active drivers to the port (may be conflicts, yet resolved value is high)
`N`	`x` input	No check	`x` input	No check	Unknown (not needed)
`n`	`x` input	No check	`x` input	No check	Unknown (not needed)
`Z`	`z` input	No check	`z` input	No check	Tri-state

output Direction

The driving direction is `output` if the driving value from drivers inside the DUT is non-tristate, but the value driven by the drivers in the test fixture is tri-state. The resolved value is mapped in Table 14-7.

Table 14-7 output Value Mapping

Port Value	Declared `in`	Declared `out`	Declared `bi` Enable = 0	Declared `bi` Enable = 1	Mapped VCD Value
`L`	`z` input	Check `0`	`z` input	Check `0`	Low – only one active driver to the port
`l`	`z` input	Check `0`	`z` input	Check `0`	Low – two or more active drivers to the port (may be conflicts, yet resolved value is low)
`H`	`z` input	Check `1`	`z` input	Check `1`	High – only one active driver to the port

Table 14-7 output Value Mapping, ***continued***

Port Value	Declared `in`	Declared `out`	Declared `bi` Enable = 0	Declared `bi` Enable = 1	Mapped VCD Value
`h`	`z` input	Check `1`	`z` input	Check `1`	High – two or more active drivers to the port (may be conflicts, yet resolved value is high)
`X`	`z` input	No check	`z` input	No check	Unknown (not needed)
`T`	`z` input	No check	`z` input	No check	Tri-state

unknown Direction

The driving direction is `unknown` if both the drivers in the test fixture and DUT are driving a non-tristate value. The resolved value is mapped in Table 14-8.

Table 14-8 unknown Value Mapping

Port Value	Declared `in`	Declared `out`	Declared `bi` Enable = 0	Declared `bi` Enable = 1	Mapped VCD Value
`0`	`0` input	Check `0`	`0` input	Check `0`	Low (`input=0` and `output=0`)
`1`	`1` input	Check `1`	`1` input	Check1	High (`input=1` and `output=1`)
`?`	`x` input	No check	`x` input	No check	`x` (`input=x` and `output=x`)
`A`	`0` input	Check `1`	`0` input	Check `1`	`x` (`input=0` and `output=1`)
`a`	`0` input	No check	`0` input	No check	`x` (`input=0` and `output=x`)
`B`	`1` input	Check `0`	`1` input	Check `0`	`x` (`input=1` and `output=0`)
`b`	`1` input	No check	`1` input	No check	`x` (`input=1` and `output=x`)

Table 14-8 unknown Value Mapping, *continued*

Port Value	Declared `in`	Declared `out`	Declared `bi` Enable = 0	Declared `bi` Enable = 1	Mapped VCD Value
`C`	`x` input	Check `0`	`x` input	Check `0`	`x` (`input=x` and `output=0`)
`c`	`x` input	Check `1`	`x` input	Check `1`	`x` (`input=x` and `output=1`)
`F`, `f`	`z` input	No check	`z` input	No check	Tri-state (`input=z` and `output=z`)

Enhanced VCD Format Example

The following is an example of EVCD file format.

```
$date
Jul 11, 2004 15:42:26
$end
$version
TOOL: ncsim 05.00-p001
$end
$timescale
1 ns
$end
$scope module board $end
$scope module counter $end
$var port [3:0] ! value $end
$var port 1  clock $end
$var port 1 # fifteen $end
$var port 1 $ altFifteen $end
$upscope $end
$upscope $end
$enddefinitions $end
#0
$dumpports
pXXXX 6666 6666 !
pN 6 6 "
pX 6 6 #
pX 6 6 $
$end
```

```
#5
pU 0 6 "
#10
pLLLL 6666 0000 !
pL 6 0 #
pL 6 0 $
#50
pD 6 0 "
...
```

Expected Output and Comparison Result Waveforms for Value Change Dump Files

If a VCD or EVCD contains output or bi-directional vectors, the Virtuoso UltraSim simulator generates two waveform files: One contains all the expected output vector waveforms as specified in the VCD or EVCD file and the other contains the waveforms from the comparison results.

You can use the following statement in the VCD or EVCD file to enable or disable the simulator from generating these waveforms (default is `1` or enabled).

```
.output_wf 0|1
```

The waveform format is defined by the `wf_format` option in the analog netlist file. A maximum of two waveform files are generated for one or more VCD or EVCD files. The expected waveform filename is `netlist.vecexp.trn` (PSF, FSDB, etc.) and the output vector is `signal_name_exp`. The comparison waveform filename is `netlist.vecerr.trn` (PSF, FSDB, etc.) and each comparison waveform is `signal_name_err`.

The comparison result values include,

0 – matched

1 – mismatched

X – ignored (output vector = X or bi-directional vector at input stage are possible causes)

In addition to the individual comparison result waveforms, the simulator generates a single `vec_error` waveform to indicate the overall comparison results. Waveform `vec_error` equals `1` when any of the individual comparison result waveforms also have a value of `1` (`X` is treated as `0`).

Frequently Asked Questions

Is it necessary to modify the VCD/EVCD file to match the signals?

You can adjust the signal information file to match signals in the VCD/EVCD file with those in the netlist file, and leave the VCD/EVCD file unchanged. The Virtuoso UltraSim simulator only needs the scopes specified in the `.scope` statement and ignores the other scopes (the simulator also ignores the parent or child scope of the specified scope). The `.alias` statement can be used to map the signal names between the VCD/EVCD and circuit netlist files.

How can I verify the input stimuli?

As digital vector format, first probe the signals in the top-level using `.probe tran v(*) depth=1` and check the waveform outputs with the Virtuoso Visualization and Analysis or SimVision viewers.

Note: The signal names are case sensitive.

Review the log file to check if the signals defined in the digital vector file match those defined in the analog netlist file.

- If the signal is in the specified scope of VCD/EVCD, but not in the VCD info file, a warning message appears.
- If the signal is in the specified scope of VCD/EVCD and in the VCD info file, but not in the analog netlist file, a warning message appears.
- If the signal is in the analog netlist file, but does not match the one in VCD/EVCD, check the list of dangling nodes or no DC path to the ground.

How do I verify the output vector check?

A `netlist.veclog` file is generated at the location specified by the Virtuoso UltraSim simulator `option-raw` statement if there are any vector checks. A `netlist.vecerr` file is also generated when errors occur during the vector check. Refer to these two files for detailed information about the vector check.

When the signal is defined in both the VCD/EVCD files and the VCD info file, but not in the analog netlist file, the simulator issues a warning message.

In addition, the simulator generates two waveform files: One contains all the expected output vector waveforms as specified in the VCD or EVCD file and the other contains the waveforms from the comparison results.

Why should I use hierarchical signal name mapping?

Flat signal name mapping works for most situations, but suffers from the following limitations:

- Only the signals in the top level can be mapped to the VCD file (analog netlist file).
- When multiple `.scope` statements are used in a digital VCD file, the Virtuoso UltraSim simulator treats them as a single set and searches for signals (as defined in the `.in`, `.out`, and `.bi` statements) in all of the `.scope` statements. An error occurs when a signal with the same name appears in more than one `.scope` statement.

Hierarchical signal name mapping is able to overcome these limitations, allowing you to map signals to the lower levels of the analog netlist file and to use multiple `.scope` statements (see "Enhanced Statements" on page 700 for more information about `.scope` statements).

What is the difference between CPU and user time?

Description

- **CPU time** is the time the central processing unit (CPU) spends running the user program
- **User time** is the user and system times combined (that is, the total time needed to provide system service to the user program)

When running a simulation, the Virtuoso UltraSim simulator stores the elapsed CPU and user time information in the STDOUT (standard output) or log file. The estimated completion time is based on linear interpolation of the user time from the start of the simulation.

Note: The elapsed user time can be less than the elapsed CPU time.

Example

```
Completed transient up to: 1.020019e-06 (60%) at Tue Sep  5 11:26:39 2006 memory:
393.1200 KB total: 63.0314 MB elapsed user time: 0:00:10 (10.240 sec), elapsed
CPU time: 0:00:10 (10.280 sec) estimated completion time: 0:00:06
**** NUM_EVENTS: 144490  ****
```

15

Flash Core Cell Models

This chapter describes the flash core cell macro models supported by the Virtuoso® UltraSim™ simulator. The simulator is able to model the floating gate effect of the flash core cell, in addition to conventional MOSFET models, such as MOS level 1 and BSIM3.

The flash core cell model is more accurate and flexible than conventional models using analog hardware description language (HDL) or complicated subcircuits comprised of many elements. The model also allows simultaneous simulation of the flash core cell with peripheral circuitry. Based on physical theory and parameterized equations, the effects of the floating gate charge are tracked by the threshold voltage (Vth) of the cell. The flash core cell can take on different events, such as programming, erasing, and reading during the simulation. Since each type of event is controlled by different physics, the Vth equations are also different for each event.

Device

```
mcell nd ng ns npw ndnw [ndnw] model_name [l=length] [w=width] [deltvthinit=val]
     [delvto=val]
```

Description

The flash core cell is a MOSFET device with a hidden floating gate. The Virtuoso UltraSim simulator supports single n-well and embedded p-well processes. A single n-well process results in a four pin device and an embedded p-well process results in a five pin device. The device parameters are listed in Table 15-1 on page 713.

Table 15-1 Device Parameters

Parameter	Description
`mcell`	MOSFET transistor for flash core cell
`nd`	Drain node
`ng`	Gate node
`ns`	Source node

Table 15-1 Device Parameters, *continued*

Parameter	Description
npw	P-well node for n-MOSFET device
ndnw	Deep N-well node (optional)
model_name	Model name
l	Device length
w	Device width
deltvthinit or delvto	Device Vth shift at time 0 (default is 0 V)

Models

```
.model model_name flashcell flashlevel=val <parameter1=val1> <parameter2=val2>
```

Note: The `flashcell` keyword is used to indicate that the model card is a flash core cell model card and it can be set to 1, 2, or 3 according to the flash cell type being simulated (model parameters for different flash cell types are listed in Tables 15-2, 15-3, and 15-4).

Description

The Virtuoso UltraSim simulator supports flash core cell models based on MOS level 1 and BSIM3v3 models. The Vth of the transistor varies according to parameterized equations. The flash core cell model consists of two parts: The flash core cell model parameters used to model Vth changes and the conventional MOSFET model.

Table 15-2 Model Parameters for flashlevel=1 (NOR Type)

Parameter	Description	Default Value
`tpgmstep`	Timestep Vth is adjusted for programming event during simulation	10 ns
`tersstep`	Timestep Vth is adjusted for erasing event	500 ns
`kpgm`	Change in Vth during programming event (per `tpgmstep`)	2e-3 1/V
`kers`	Change in Vth during erasing event (per `tersstep`)	0.15e-3 1/V
`vgpgmmin`	Minimum of Vgate during programming event	1 V
`vdpgmmin`	Minimum of Vdrain during programming event	1 V
`vspgmmax`	Maximum of Vsource during programming event	1 mV
`vpwpgmmax`	Maximum of Vpwell during programming event	0 V
`vgersmax`	Maximum of Vg during erasing event	0 V
`vpwersmin`	Minimum of Vpwell during erasing event	0 V
`vdersmin`	Minimum of Vdrain during erasing event	No default
	Note: The `vdersmin` parameter is a required flash core cell model card parameter.	
`vsersmin`	Minimum of Vsource during erasing event	0 V
`flashvcc`	Flash core cell voltage supply	20 V
`vthigh`	Absolute maximum of Vth	10 V
`vtlow`	Absolute minimum of Vth	10 V
`vtpgmmaxshift`	Absolute maximum Vth shift in a single programming event	10 V
`vtersmaxshift`	Absolute maximum Vth shift in a single erasing event	10 V

Table 15-3 Model Parameters for flashlevel=2 (NOR Type)

Parameter	Description	Default Value
kpgm	Change in Vth during programming event (per tpgmstep)	2e-3 1/V
kers	Change in Vth during erasing event (per tersstep)	0.1e-3 1/V
tpgmstep	Time interval for Vth update during programming event	10 ns
tersstep	Time interval for Vth update during erasing event	500 ns
vgpgmmin	Minimum gate voltage to start programming event	1 V
vgpgmmax	Maximum gate voltage to start programming event	10 V
vdpgmmin	Minimum drain voltage to start programming event	1 V
vdpgmmax	Maximum drain voltage to start programming event	10 V
vpwpgmmin	Minimum pwell voltage to start programming event	-4 V
vpwpgmmax	Maximum pwell voltage to start programming event	0 V
vnwpgmmin	Minimum nwell voltage to start programming event	0 V
vnwpgmmax	Maximum nwell voltage to start programming event	20 V
vgersmin	Minimum gate voltage to start erasing event	-15 V
vgersmax	Maximum gate voltage to start erasing event	0 V
vdersmin	Minimum drain voltage to start erasing event	0 V
vsersmin	Minimum source voltage to start erasing event	0 V
vsersmax	Maximum source voltage to start erasing event	1 V
Vpwersmin	Minimum pwell voltage to start erasing event	0 V
vnwersmin	Minimum nwell voltage to start erasing event	3.5 V
vtpgmmaxshift	Maximum Vth shift during programming event	10 V
vtersmaxshift	Maximum Vth shift during erasing event	10 V
vthigh	Maximum value of Vth after programming event	10 V
vtlow	Maximum value of Vth after erasing event	-10 V

Table 15-4 Model Parameters for flashlevel=3 (NAND Type)

Parameter	Description	Default Value
kpgm	Change in Vth during simulation programming event (per tpgmstep)	1V/1 us
kers	Change in Vth during erasing event (per tersstep)	1V/1 us
tpgmstep	Time interval for Vth update during programming event	10 ns
tersstep	Time interval for Vth update during erasing event	10 ns
vgspgmmin	Minimum gate to source voltage to start programming event	10 V
vgspgmmax	Maximum gate to source voltage for programming event	30 V
vsbpgmmax	Maximum source-to-body voltage to start program	0.5 V
vpwgersmin	Minimum pwell to gate voltage to start erasing event	10 V
vpwgersmax	Maximum pwell to gate voltage for erasing event	30 V
delvto	Initial value for cell Vth	0 V
vthigh	Maximum value of Vth after programming event	5 V
vtlow	Maximum value of Vth after erasing event	-5 V

Examples

The flash core cell model must be included with the conventional MOSFET model in order to function. To include a MOSFET model card in a flash core cell model, use the following command:

```
.appendmodel flash=dest_mod_name model=src_mod_name
```

The name of the flash core cell model card is `dest_mod_name` and `src_mod_name` is the name of the conventional MOSFET model card.

For example

```
.model fnmos1 flashcell flashlevel=1 vthigh=10
```

tells the simulator that `fnmos1` is a flash core cell model and that `flashlevel` and `vthigh` are its model parameters.

The next example

```
.model tn nmos level=49 vtho=0.0 k1=0.4 k2=0.3
.model nandcell flashcell flashlevel=3 vthigh=10 kpgm=2m vgspgmmin=10
.appendmodel flash=nandcell model=tn
mcell d g s pw tn l=0.9 w=1 delvto=-0.7
```

tells the Virtuoso UltraSim simulator that `nandcell` is a flash core cell model and `tn` is a conventional MOSFET model on which the flash core cell model is attached.

16

VST/VAVO/VAEO Interfaces

This chapter describes the Virtuoso® VoltageStorm Transistor (VST), Virtuoso Analog VoltageStorm Option (VAVO), and Virtuoso Analog ElectronStorm Option (VAEO) interfaces for the Virtuoso UltraSim™ simulator.

VST Interface

The Cadence VoltageStorm Transistor-Level PGS tool is used to analyze the power distribution network for IR voltage drop and metal electromigration failure for a circuit design. The Virtuoso UltraSim simulator supports the following flows:

- Lumped capacitance in signal nets for higher performance in actual circuit simulation.
- Distributed resistance and capacitance in signal nets for greater signal timing accuracy.

The Virtuoso UltraSim `usim_ir` command is designed to be used with these flows. For more information about this tool, refer to the *VoltageStorm Transistor-Level PGS User Guide*.

VAVO/VAEO Interface

Spectre Syntax

```
usim_emir type=all format=[vavo] [start=time] [stop=time]
```

SPICE Syntax

```
.usim_emir type=all format=[vavo] [start=time] [stop=time]
```

Note: A period (`.`) is required when using SPICE language syntax (for example, `.usim_emir`).

Description

To improve the efficiency and capability of the Virtuoso UltraSim simulator and VAVO/VAEO flow, the simulator calculates the information needed for electromigration (EM) and IR drop analysis, including maximum, root mean square (RMS), and average voltage values for each node, and average current values for each resistor. The information is saved in a binary database that can be read into VAVO/VAEO for post-processing.

The `usim_emir` command can be added to the netlist file, so that the Virtuoso UltraSim simulator saves the binary database, and VAVO/VAEO continues to run uninterrupted.

Arguments

`format`	The Virtuoso UltraSim simulator saves the voltage and current information in a binary database (`vavo` keyword is specified for Virtuoso UltraSim and VAVO/VAEO flow). **Note:** The `usim_emir` command can also be used with the Virtuoso UltraSim netlist-based EM/IR flow (see Chapter 9, “Netlist-Based EM/IR Flow” for more information).
`start`/`stop`	Specifies the time window start and stop times. The `start` time default is the beginning of the transient simulation and the `stop` time default is the end of the transient simulation.

A

Virtuoso UltraSim L/XL Product Level Comparison Table

The following table lists all of the Virtuoso UltraSim L and XL product level features.

Feature	UltraSim L	UltraSim XL
High Level Options		
sim_mode	**X**	**X**
UltraSim L: `s`, `a`, `ms`, `da`, and `df` modes		
UltraSim XL: `s`, `a`, `ms`, `da`, `df`, and `dx` modes		
analog	**X**	**X**
speed	**X**	**X**
Solver Options		
tol	**X**	**X**
method	**X**	**X**
trtol	**X**	**X**
maxstep_window	**X**	**X**
gmin_allnodes	**X**	**X**
cmin_allnodes	**X**	**X**
Device Model Options		
mosd_method	**X**	**X**
diode_method	**X**	**X**
vdd	**X**	**X**
deg_mod	**X**	**X**

Feature	UltraSim L	UltraSim XL
minr	X	X
RC Reduction		
postl		X
preserve		X
rcr_fmax		X
rshort	X	X
rvshort	X	X
lshort	X	X
lvshort	X	X
cgnd	X	X
cgndr	X	X
DC Options		
dc	X	X
dc_exit	X	X
dc_prolong	X	X
abstolv	X	X
abstoli	X	X
Miscellaneous		
ade	X	X
rcut	X	X
canalog	X	X
canalogr	X	X
hier_delimiter	X	X
duplicate_subckt	X	X
strict_bin	X	X
buschar	X	X
progress_t	X	X

Feature	UltraSim L	UltraSim XL
progress_p	X	X
vl	X	X
vh	X	X
sim_start	X	X
dump_step	X	X
hier	X	X
pa_elemlen	X	X
Warning Message Control		
warning_limit	X	X
warning_limit_dangling	X	X
warning_limit_floating	X	X
warning_limit_near_float	X	X
warning_limit_ups	X	X
Waveform Output		
wf_format	X	X
wf_maxsize	X	X
wf_reltol	X	X
wf_tres	X	X
wf_abstolv	X	X
wf_abstoli	X	X
wf_filter	X	X
Stitching/Backannotation		
capfile		X
spef		X
spf		X
dpf		X
cmin		X

Feature	UltraSim L	UltraSim XL
cmingnd		X
cmingndratio		X
dpfscale		X
spfbusdelim		X
spfcaponly		X
spfcrossccap		X
spffingerdelim		X
spfhierdelim		X
spfinstancesection		X
spfkeepbackslash		X
spfnamelookup		X
spfrcreduction		X
spfrecover		X
spfscalec		X
spfscaler		X
speftriplet		X
spfxtorprefix		X
rmin		X
rvmin		X
Voltage Regulator		
.usim_vr		X
Power Network Solver		
.usim_pn		X
.usim_ups		X
Interactive Mode		
-i	X	X
HSPICE		

Feature	UltraSim L	UltraSim XL
.meas	X	X
.probe	X	X
bisection		X
Dynamic Checks		
.acheck		X
.dcheck		X
.pcheck		X
.usim_nact	X	X
.usim_pa	X	X
.usim_ta	X	X
Static Checks		
.usim_report capacitor		X
.usim_report chk_maxleak		X
.usim_report chk_mosv		X
.usim_report chk_nmosb		X
.usim_report chk_nmosvgs		X
.usim_report chk_param		X
.usim_report chk_pmosb		X
.usim_report chk_pmosvgs		X
.usim_report chk_substrate		X
.usim_report node	X	X
.usim_report param	X	X
.usim_report partition	X	X
.usim_report resistor		X
Fast Envelope		
env_clockf		X
env_method		X

Feature	UltraSim L	UltraSim XL
env_nsamples		X
env_maxnstep		X
env_tstart		X
env_tstop		X
env_tol		X
env_trtol		X
env_speed		X
env_harms		X
env_resolve		X
env_ignore_digital		X
Reliability		
.age		X
.agemethod		X
.ageproc		X
.deltad		X
.hci_only		X
.minage		X
.nbti_only		X
.nbtiageproc		X
Vector Stimuli		
.vec	X	X
.vcd	X	X
Flash Model		
.appendmodel		X
EMIR		
.usim_emir		X
VST Dynamic Flow		

Feature	UltraSim L	UltraSim XL
.usim_ir	X	X

B

Reader Survey

In an effort to continuously improve the Virtuoso® UltraSim™ documentation, Cadence Technical Publications has created a survey to collect information from readers regarding the *Virtuoso UltraSim Simulator User Guide*.

Please take a few minutes to respond to the survey. All information submitted is confidential and will only be used by Cadence Technical Publications to improve the Virtuoso UltraSim documentation.

You can submit the completed survey using any one of the following methods:

- Cut/paste the survey from your browser window into an email and send to:

  `schwirzk@cadence.com`

- Print out the survey and mail it to:

  ```
  Cadence Design Systems, Inc.
  c/o Martin Schwirzke
  555 River Oaks Parkway
  San Jose, CA 95134
  USA
  ```

- Hand the survey to a Cadence UltraSim Application Engineer (AE) or Product Engineer (PE).

1. **Which version of the Virtuoso UltraSim Simulator User Guide do you currently use? (Check one).**

   ❑ MMSIM 6.1 ❑ MMSIM 6.2 ❑ MMSIM 7.0 ❑ Other: ______________

2. **What type of information do you look for "most" in the manual?**

   ____

   ____

   If you need more space to write your response to a question, use the back of this survey.

3. **In general, are you able to find the information?** Yes ___ No ___

   If you answered "No," why not? ______________________________________

   ____

   ____

4. **Is the information helpful?** Yes ___ No ___

   If you answered "No," why not? ______________________________________

   ____

   ____

5. **Rate the "organization" of information in the manual (circle one)**.

Poor		Adequate		Excellent
1	2	3	4	5

6. **Rate the overall "quality" (completeness and accuracy) of the manual**.

Poor		Adequate		Excellent
1	2	3	4	5

7. **What do you like "best" about the manual?**

   ____

   ____

8. **What do you like "least" about the manual?**

   ____

____

9. **What is the "one" most important thing Cadence can do to improve the manual?**

____

____

10. **Write any other comments about the *Virtuoso UltraSim Simulator User Guide* below.**

____

____

____

Can we contact you regarding your comments? Yes ___ No ___

If "Yes," please provide your contact information below:

____

____

____

____

Index

Symbols

A

B

C

D

E

F

G

N

O

P

R

S

V

W

X