cadence®

Spectre FX Circuit Simulator User Guide

Product Version 23.1
September 2023

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

Preface

The demand for performance, accuracy, and capacity from transistor-level simulators continues to rise at an increasing pace. On the other hand, the advancement of process nodes, the adoption of new design practices, the need for full-chip simulation, and the stronger effect of parasitic elements greatly increase the size and complexity of the designs to be simulated. Spectre® FX is the next-generation FastSPICE simulator from Cadence that addresses such simulation challenges.

The following are the highlights of the Spectre FX simulator:

- Common Spectre front-end for parsing, device models, and infrastructure support
- Built-in SPICE solver that serves as a strong foundation for accuracy in results
- FastSPICE technology innovations that provide better performance, accuracy, and higher capacity
- Out-of-the-box yet simpler use model
- Comprehensive FastSPICE analysis features including back-annotation and circuit checks

Licensing

Cadence offers both ala carte pricing plus a unique multi-mode simulation (MMSIM) license that can enable the Spectre simulation platform. Spectre circuit simulator, Spectre APS, Spectre® XPS, Spectre® X, and Spectre® FX are the base products. This section lists the features of Spectre FX and the options available with this simulator.

Both Spectre FX and its options are accessible using Spectre MMSIM tokens. Contact Cadence Customer Support for details.

Table -1 Spectre FX Features

Unlimited element count
Circuit checks

Mixed signal CoSim
EMIR

Table -2 Options for Spectre FX

Spectre Option	Description
Spectre FX Multi-Core Simulation Option	Enables multi-core simulation up to 64 cores with the Spectre FX base product on a single machine. To run multi-threading on 4 cores, you need 1 Spectre FX Multi-Core Simulation Option. You need 2 Spectre FX Multi-Core Simulation Options to perform multi-threading computation on 16 cores. To run multi-threading on 64 cores, you need 3 Spectre FX Multi-Core Simulation Options.
Spectre Power option	Enables EMIR analysis with Spectre X base product. It also enables EMIR analysis with Spectre APS base product and the Spectre CPU Accelerator option. This option is not available with Spectre base product.

Related Documents for Spectre FX

The following documents provide more information about Spectre FX and related products.

- The Spectre FX circuit simulator shares the front end and infrastructure with the Spectre circuit simulator. For more information on the shared features, refer to *Spectre Classic Simulator, Spectre APS, Spectre X, and Spectre XPS User Guide*.
- To learn more about the specific parameters of components and analyses, refer to the *Spectre Circuit Simulator Reference* manual.
- To learn more about the equations used in the Spectre circuit simulator, refer to the *Spectre Circuit Simulator Components and Device Models Reference* manual.
- The Virtuoso Visualization and Analysis tool is used to display the simulation results. For more information about the tool, see *Virtuoso Visualization and Analysis User Guide*.

- For more information about using the Spectre circuit simulator with Verilog-A, see the *Cadence Verilog-A Language Reference* manual.
- The Spectre circuit simulator is often run within the analog circuit design environment, under the Cadence design framework II. To see how the Spectre circuit simulator is run under the analog circuit design environment, read the *Virtuoso ADE Explorer User Guide*.
- For more information about how you work with the design framework II interface, see *Cadence Design Framework II Help*.
- For more information about specific applications of the Spectre analyses, see *The Designer's Guide to SPICE & Spectre*[1].

Typographic and Syntax Conventions

This list describes the syntax conventions used for the Spectre circuit simulator.

`literal`	Nonitalic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names, filenames and paths, and any other sort of type-in commands.
`argument`	Words in italics indicate user-defined arguments for which you must substitute a name or a value. (The characters before the underscore (_) in the word indicate the data types that this argument can take. Names are case sensitive.
|	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.
`[ ]`	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.
`{ }`	Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.
`...`	Three dots (...) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.

1. Kundert, Kenneth S. *The Designer's Guide to SPICE & Spectre.* Boston: Kluwer Academic Publishers, 1995.

1

Introduction to the Spectre FX Circuit Simulator

Spectre® FX simulator is the next-generation FastSPICE simulator that meets the increasing demand for performance, accuracy, and capacity from transistor-level simulators needed for complex and full-chip designs.

Spectre FX shares a common front-end and infrastructure with the other simulators in the Spectre platform. It provides support for the native Spectre syntax and is fully compatible with HSPICE® (from Synopsys, Inc.) netlist files. This built-in SPICE solver provides a strong foundation for accuracy in results.

Related Topics

Features of Spectre FX

Starting a Spectre FX Simulation Run

Command-Line Options for Spectre FX

Features of Spectre FX

Spectre FX focuses only on the transient analysis. The following table lists the features supported by Spectre FX:

Feature	Supported	Note
SPICE Syntax	Yes	
SPICE-Compatible Options	Partial	Supports only a few options for case setup, such as gmin, parhier, scale, scalm, geoshrink, measdgt, tnom, search, and autostop.
SPICE-Compatible Controls	Partial	Supports only a few controls for case setup, such as .alter, .data, .tran, .measure, .probe or.print, .ic, .connect, .lib, .subckt or .macro/.ends or .eom, .if and .else.
Spectre Syntax	Yes	
Device Models	Yes	Shares the same infrastructure as Spectre APS.
Verilog-A	Yes	
Digital Vector and VCD files	Yes	
Backannotation	Yes	See EMIR Analysis and Post-Layout Simulations for more information.
Probing and Measuring	Yes	Supports .probe or.print/.measure in SPICE syntax and `save` in Spectre syntax. `.print` is converted to `.probe`.
Autostop of Measurement	Yes	Enabled using the option autostop. By default, it is disabled.
Analyses	Partial	Supports DC, AC, transient, monte carlo, and reliability analyses.
Alter and Sweep Analyses	Limited	Only the basic flow is supported. Both SPICE and Spectre syntaxes are supported.
Save and Recover	Yes	Supports the same syntax as other Spectre simulators.
Circuit Checks	Yes	See Circuit Checks for more information.
Monte Carlo Analysis	Yes	See Monte Carlo Analysis for more information.

Feature	Supported	Note
Bisection Optimization	No	
IR drop and EM Analysis	No	
Virtuoso® ADE Product Suite	Yes	

Starting a Spectre FX Simulation Run

To start a Spectre FX simulation run:

➡ Type the following at the command line:

```
spectrefx [+preset=mode] +[option] [+spectre] [-f fsdb] [+mt=N] [-outdir run1]
[-outname SpectreFX.out].... top.sp
```

Examples

```
% spectrefx -f fsdb +mt=4 top.sp
% spectrefx -f fsdb +mt=8 +preset=mx top.sp
% spectrefx -f fsdb +mt=8 +speed=ax +optimize=mx top.sp -outdir run1
```

Spectre FX automatically sets spice or spectre compatible mode by detecting the input file format and prints an information message in the log file. However, you can add `+spice` or `+spectre` at the command line to ensure the compatible mode is used as you expect.

Related Topics

Command-Line Options for Spectre FX

Modes Compatible with Spectre FX

Multithreading in Spectre FX

Output Files of a Spectre FX Run

Managing Output Files Using Predefined Percentage Codes

Reviewing the Log File

Command-Line Options for Spectre FX

The following table describes the command-line options that can be used for Spectre FX simulator:

Option	Description
`spectrefx`	Enables the Spectre FX simulator.
`-help`	Lists the Spectre FX command options and their descriptions. You can use `-h` as an abbreviation of `-help`.
`-V`	Prints the Spectre version information.
`-W`	Prints the sub-version information.
`+spectre`	Ensures Spectre compatibility.
`+spice`	Ensures SPICE compatibility.
`[+preset=`*`mode`*`]`	Specifies the preset mode. Possible values are `baseax`, `basemx`, `baselx`, `basevx`, `ax`, `mx`, `lx`, and `vx`. The default mode is `lx`.
`+optimize=`*`value`*	Defines the parasitic optimization in Spectre FX. The most accurate parasitic optimization can be obtained using the value `ax`. The highest parasitic optimization can be obtained using the value `vx`. Possible values are `ax`, `mx`, `lx`, and `vx`.
`+speed=`*`value`*	Defines the simulation tolerance for voltage and current calculation. Possible values are `ax`, `mx`, `lx`, and `vx`. The default value is determined by the preset mode.
`+multithread=`*`N`* `+mt=`*`N`*	Turns on the multithreading capability. *`N`* is the specified number of threads, up to a maximum of 64 threads. `+mt` can be used as an abbreviation of `+multithread`. By default, multithreading is disabled in Spectre FX. You can use `+mt` as an abbreviation of `+multithread`. This option also lets you run simulations in distributed mode with LSF. Related topic: Multithreading in Spectre FX
`+config` *`file`*	Enables you to append all the Spectre FX commands defined in *`file`* to the netlist. *`file`* may contain one or more Spectre FX netlist lines. Multiple `+config` statements are supported on the same Spectre FX command line and are processed incrementally.

`+pre_config` *`file`*	Enables you to prepend all Spectre commands defined in *`file`* to the netlist. *`file`* may contain one or more Spectre netlist lines. Multiple `+pre_config` statements are supported on the same Spectre command line and are processed incrementally.
`+echooptions`	Dumps the options specified in the netlist and configuration files.
`-format` *`fmt`*	Generates the raw data in the specified format. Possible values are `psfbin`, `psfascii`, `psfbinf`, `psfxl`, `sst2`, and `fsdb`. You can use `-f` as an abbreviation of `-format`.
`-outdir` *`path`*	Changes the default location of the Spectre FX output files. However, this option neither changes the location of the raw directory if explicitly specified by using the -raw option, nor changes the location of the files that have slashes in their names. You can use `-o` as an abbreviation of `-outdir`.
`+top` *`value`*	When you specify `+top` with a subcircuit name, all elements, models, and so on, will be exposed to the top level and the subcircuit will not exist anymore. As a result, instantiation from this subcircuit will be illegal. Instances inside the subcircuit will automatically connect to the instance at the top level, if they connect to the same node.
`-cmiversion`	Prints CMI version information.
`-cmiconfig` *`file`*	Reads the specified file for information to modify the existing CMI configuration.
`-dynchecks`	Disables the dynamic checks capability.
`-log`	Displays the log information on the standard output (shell) only and does not copy it to a log file. You can use `-l` as an abbreviation of `-log`.
`+log` *`file`*	Displays the log information on the standard output (shell) and copies it to the specified log *`file`*. You can use `+l` as an abbreviation of `+log`.
`=log` *`file`*	Sends the log information to the specified *`file`* only and does not display it on the standard output (shell). You can use `=l` as an abbreviation of `=log`.
`-raw` *`raw`*	Saves the simulation results in the specified file or directory. If `%C` is specified in the file or directory name, it is replaced by circuit name. You can use `-r` as an abbreviation of `-raw`.

`+savestate`	Enables the savestate capability. You can use `+ss` as an abbreviation of `+savestate`.
`-savestate`	Disables the savestate capability. You can use `-ss` as an abbreviation of `-savestate`.
`+recover[=`*`filename`*`]`	
`+rec[=`*`filename`*`]`	Restarts the simulation using a checkpoint or savestate file. If both checkpoint and savestate files exist and the file name is not specified, the savestate file is used. You can use `+rec` as an abbreviation of `+recover`.
`-recover`	Does not restart the simulation even if a checkpoint file exists. You can use `-rec` as an abbreviation of `-recover`.
`-cols` *`N`*	Sets the screen width based on the number of characters specified. This is needed only if the simulator cannot determine the screen width automatically or if the default value of 80 is not acceptable. You can use `-c` as an abbreviation of `-cols`. **Note:** Spectre FX cannot determine the screen width if the output is redirected to a file or a pipe.
`-colslog` *`N`*	Sets the log file width based on the number of characters specified. The default log file width is 80 characters.
`+error`	Prints the error messages.
`-error`	Does not print the error messages.
`+warn`	Prints the warning messages on the screen.
`-warn`	Does not print the warning messages on the screen.
`-maxwarns` *`N`*	Specifies the maximum number of times a particular type of warning message will be issued per analysis. You can use `-maxw` as an abbreviation of `-maxwarns`.
`-maxnotes` *`N`*	Specifies the maximum number of times a particular type of notice message will be issued per analysis. You can use `-maxn` as an abbreviation of `-maxnotes`.
`-maxwarnstolog` *`N`*	Specifies the maximum number of times a particular type of warning message will be printed to the log file per analysis. You can use `-maxwtl` as an abbreviation of `-maxwarnstolog`.

`-maxnotestolog` *N* `-maxnt` *N*	Specifies the maximum number of times a particular type of notice message will be printed to the log file per analysis. You can use `-maxntl` as an abbreviation of `-maxnotestolog`.
`+note`	Prints the notice messages on the screen.
`-note`	Does not print the notice messages on the screen.
`+info`	Prints the informational messages.
`-info`	Does not print the informational messages.
`+lqtimeout <`*value*`>`	Specifies the duration (in seconds) for which Spectre FX should wait to retrieve a license. When set to `0`, Spectre FX waits until the license is available. You can use `+lqt` as an abbreviation of `+lqtimeout`.
`+lqsleep <`*value*`>`	Specifies the wait duration between two attempts made by Spectre FX to check out a license when queuing. Setting the value to a positive number overrides the default wait duration of 30 seconds. You can use `+lqs` as an abbreviation of `+lqsleep`.
`+lsuspend`	Enables the license suspend/resume capability. When Spectre FX receives `SIGTSTP`, it checks in all the licenses before it gets suspended. The licenses are checked out again when `SIGCONT` is received. You can use `+lsusp` as an abbreviation of `+lsuspend`.
`+liclog`	Writes the license check-in and check-out information in the log file.

Related Topics

Multithreading in Spectre FX

Output Files of a Spectre FX Run

Managing Output Files Using Predefined Percentage Codes

Modes Compatible with Spectre FX

Spectre FX automatically sets spice or spectre compatible mode by detecting the input file format and prints an information message in the log file. However, you can add `+spice` or `+spectre` at the command line to ensure the compatible mode is used as you expect.

- `+spice` ensures that the netlist convention and device models are considered to be in SPICE format
- `+spectre` ensures that the netlist convention and device models are considered to be in Spectre format.

The following examples show the log file entries with different command-line options:

- `% spectrefx -f fsdb +mt=4 top.sp`

  Comment in the log file:

  ```
  Enabled Spice mode for Spectre FX.
  ```

- `% spectrefx -f fsdb +mt=4 +spice top.sp`

  Comment in the log file:

  ```
  Enabled Spice mode for Spectre FX based on the command-line option +spice.
  ```

- `% spectrefx -f fsdb +mt=4 top.scs`

  Comment in the log file:

  ```
  Enabled Spectre mode for Spectre FX.
  ```

- `% spectrefx -f fsdb +mt=4 +spectre top.scs`

  Comment in the log file:

  ```
  Enabled Spectre mode for Spectre FX based on the command-line option +spectre.
  ```

Note: If you do not specify `+spice` or `+spectre` at the command line, Spectre FX checks the environment variable `SPECTRE_COMPATIBLE_FX`. If `SPECTRE_COMPATIBLE_FX` is set, Spectre FX enables Spectre compatible mode. This environment variable is ignored if you specify `+spice` or `+spectre` at the command line.

Multithreading in Spectre FX

Spectre FX supports multithreading on multi-core computer platforms. The default behavior is to use a single thread. You can enable the multithreading feature by using the `+mt=`*N* command-line option to specify the number of threads to be used, as shown below.

```
spectrefx +mt=8 ...
```

Spectre FX enables High Performance Computing (HPC) by default, therefore, it automatically adds multithreading jobs to the CPU cores from the same socket. This enhances the performance of multithreading.

When multithreading and HPC are enabled, the Spectre FX log file displays the following information:

```
HPC is enabled
Binding to socket 1
8 threads are started.
```

Enabling Distributed Mode with Multithreading

If multiple computer hosts are available, you can use the LSF model to run multi-core simulations on multiple hosts by using the `+mt` option to enable distributed mode with multithreading.

The following examples shows that each job runs with `mt=4` on one host:

```
%bsub -R "(OSNAME==Linux) span[hosts=1]" -n 4 "spectrefx +mt=lsf"
```

Related Topics

Command-Line Options for Spectre FX

Output Files of a Spectre FX Run

After the simulation completes, Spectre FX writes the following output files based on the design setup:

- Log file: Defined using the `+log`/`=log`/`-log` command-line options
- Measure results: `*.measure` and `*.mt0` files
- Waveform files and formats: Defined using the `-format` (or `-f`) command-line option
- Operating point files: `*.ic0` or `spectre.ic`
- Circuit check results

Related Topics

Using the Command-Line Option -outdir to Specify the Output Directory

Using the Command-Line Option -outname to Specify the Names of the Output Files

Managing Output Files Using Predefined Percentage Codes

Reviewing the Log File

Using the Command-Line Option -outdir to Specify the Output Directory

You can use the `-outdir` command-line option to specify the output directory for the simulation files.

The following example demonstrates the location and names of the output files.

```
% spectrefx +lx -f fsdb -outdir SFX_out top.sp
```

The following are the filenames and their locations:

```
SFX_out/
|-- top.fsdb
|-- top.log
|-- top.mt0
|-- top.measure
|-- top.raw
    |-- logFile
    |-- transient1.meas_tran
    |-- mmcheck2html.xsl
    |-- top.dynamic.xml
    |-- top.dynamic.rpt
```

Related Topics

Using the Command-Line Option -outname to Specify the Names of the Output Files

Output Files of a Spectre FX Run

Using the Command-Line Option -outname to Specify the Names of the Output Files

You can use the command-line option `-outname` to change the names of the output files and their locations. When you use this option, the waveform files are moved from the raw directory to the specified output directory. The following is the syntax to specify the `-outname` command-line option:

```
-outname out_dir_name/new_file_name
```

Here, *`out_dir_name`* is the name of the output directory and *`new_file_name`* is the name of the new file. You can specify either of the two options, or both the options in a single statement.

The following are some of the examples of using the `-outname` command-line option:

Example 1

```
% spectrefx +lx -f fsdb -outname new_outname top.sp
```

In the above example, only the name is specified with the `-outname` command-line option. This will cause the filename of the output files to change to the new one and the waveform files to move outside the raw directory. The filenames and their locations are changed as follows:

```
./
|-- new_outname.fsdb
|-- new_outname.log
|-- new_outname.mtO
|-- top.raw
```

Example 2

```
% spectrefx +lx -f fsdb -outname SFX_out/ top.sp
```

In the above example, the value `SFX_out/` of the command-line option `-outname` contains a slash (`/`), which means it is a directory name. Specifying this option causes the waveform files to move outside the raw directory.

The filenames and their locations are changed as follows:

```
SFX_out/
|-- top.fsdb
|-- top.log
|-- top.mtO
|-- top.raw
```

Example 3

```
% spectrefx +lx -f fsdb -outname SFX_out/new_outname top.sp
```

In the above example, both the output directory and the output file names are specified with the `-outname` command-line option. The filenames and their locations are changed as follows:

```
SFX_out/
|-- new_outname.fsdb
|-- new_outname.log
|-- new_outname.measure
|-- new_outname.mtO
|-- top.raw
    |-- logFile
    |-- transientl.meas  tran
    |-- mmcheck2html.xsl
    |-- traverse.css
    |-- sorttable.js
    |-- top.sqldb
    |-- top.dynamic.xml
    |-- top.dynamic.rpt
```

Related Topics

Using the Command-Line Option -outdir to Specify the Output Directory

Output Files of a Spectre FX Run

Managing Output Files Using Predefined Percentage Codes

The Spectre FX filename specification feature helps you manage your data by letting you systematically specify or modify the filenames. You can easily identify data from multiple simulation runs or from single runs containing similar repeated analyses. You can modify the input filenames to easily identify the output file from a specific simulation or analysis. You can also construct output filenames in ways that prevent accidental overwriting of data.

Spectre FX helps you keep track of the simulation data by letting you create the filenames that are variants of the input filenames. For example, with Spectre FX, you can:

- Identify the simulation data by date, time, process ID, or other defining characteristics in the results filenames.
- Keep multiple circuits in a single directory without having subsequent simulations overwrite the previous results. To do this, set the environment variables such that the output filenames are different variants of the input filenames automatically.

Creating Filenames by Modifying Input Filenames

Spectre FX supports predefined percent codes you can use in your filenames. These predefined codes let you construct filenames that add the defining characteristics, such as date or time, to the input filenames. You specify the predefined percent codes with a percent character (`%`) followed by an uppercase letter. The uppercase letter tells Spectre FX how to construct the filename. You can use the percent codes in the environment variables, command parameters, or the netlist wherever you need to specify filenames for the simulation results.

For example, `%C` is the predefined percent code for the name of the input circuit file. If your circuit file is named `opamp1` and you specify the following `-raw` setting in the UNIX environment variable, `setenv SPECTRE_DEFAULTS "-raw %C.raw"`, Spectre FX sends the simulation results to the directory named `opamp1.raw`.

Predefined Percent Codes

These are the percent code that can help you organize your simulation data:

Percentage Code	Description
`%A`	`%A` is replaced by the name of the current analysis that is running. If it is specified in a device statement, it is expanded to a blank string because there is no current analysis.
`%B`	`%B` is replaced by `64bit` for 64-bit version of the software.
`%C`	`%C` is replaced by the input circuit filename, as it is used in the command line. If the circuit file-name is `opamp1`, the specification `%C.raw` generates a file named `opamp1.raw`. When the Spectre simulator does not know the name of the input file, as when the circuit is read from the standard input (from a pipe or from a redirected file), the Spectre simulator substitutes the name `stdin` for `%C`.
`%D`	`%D` is replaced by the date when the program started. For example, the specification `%D.opamp1` might generate a file named `94-09-19.opamp1`. The date is in year-month-day format. All leading zeros are included. This format generates filenames that you can sort alphabetically into chronological order.
`%F`	`%F` is replaced by the completed analysis name. For example, if you have file named `mc1-005_dc1`, the specification `spectre_%F.dc` will expand to `spectre_mc1-005_dc1.dc`.
`%M`	`%M` is replaced by the current CMI version.
`%P`	`%P` is replaced by the process ID. The process ID is a unique integer assigned to the Spectre process by the operating system.

Percentage Code	Description
`%S`	`%S` is replaced by the simulator name. For example, the specification `%S.opamp1` might generate a file named `spectre.opamp1`. If you use a different name or a symbolic link with a different name to access a copy of the executable program, the new name becomes the program name.
`%T`	`%T` is replaced by the time when the program started. For example, the specification `%T.opamp1` might generate a file named `14:44:07.opamp1`. Time is in 24-hour format, and all leading zeros are included. This format generates filenames that sort alphabetically into a chronological order.
`%V`	`%V` is replaced by the simulator version string. For example, the specification `out_%V.raw` might generate a file named `out_1.0.2.raw`.
`%I`	`%I` is replaced by the installation directory.
`%O`	`%O` is replaced by the operating platform.
`%U`	`%U` is replaced by the user name.
`%%`	This specifies the `%` character by itself. This option lets you use percent characters in filenames. Two percent characters (`%%`) in a filename specification produce a single percent character in the filename, which is not interpreted as a percent code indicator.

The predefined percent codes feature does not perform recursive substitutions. For example, if you have an input file named `A%SD.xyz` and you create an output file with the percent code designation `%C.raw`, the Spectre simulator creates the output file `A%SDxyz.raw`. The Spectre simulator does not substitute the simulator name for `%S` in this case.

Example

For a netlist, you may run different modes in the same directory with the same output directory. You can use the predefined percent codes to avoid overwriting the output files, as shown below.

```
%> spectrefx input.sp -o input.sp_%H_%P
```

```
%> spectrefx +preset=mx input.sp -o input.sp_%H_%P
%> spectrefx +preset=ax input.sp -o input.sp_%H_%P
```

Here `%H` is the hostname and `%P` stands for process id (PID). Since each run is with different PID, the output directories are not overwritten.

Creating Filenames from Parts of Input Filenames

Colon modifiers (`:x`) create filenames from parts of input filenames. You can use colon modifiers with all percent codes except the `%%` code.

For example, if you apply `%C:r.raw` to the input filename `opamp.ckt`

where:

`%C` is the input filename (`opamp.ckt`)

`:r` is the root of the input filename (`opamp`)

`.raw` is the new filename extension

The result is the output filename `opamp.raw`. In this example, (`:r`) is the colon modifier.

Definitions of Colon Modifiers

Spectre FX recognizes the following colon modifiers:

Modifier	Description
`:r`	Signifies the root (base name) of the given path for the file.
`:e`	Signifies the extension for the given path of the file.
`:h`	Signifies the head of the given path for any portion of the file before the last `/`.
`:t`	Signifies the tail of the given path for any portion of the file after the last `/`.
`::`	Signifies the (`:`) character itself. Use two consecutive colons (`::`) to place a single colon in an output filename that is not read as a percent code modifier.

Any character except a modifier after a colon (`:`) signals the end of modifications. The Spectre FX simulator appends both the colon and the character to the filename.

Spectre FX applies a chain of colon modifiers in the sequence you specify them.

For example, if you apply `%C:e.%C:r:t` to the input filename `/circuits/opamp.ckt`

`%C:e` is the extension (`ckt`) of the input filename

`%C:r` is the root (`/circuits/opamp`) of the input filename

`:t` is the tail of the input filename after the last `/` (`opamp.ckt`)

The result is the output filename `ckt.opamp`.

Note: If the input filename does not contain a slash and a period, the modifiers `:t` and `:r` return the whole filename, and the modifier `:h` returns a period.

Examples of Colon Modifier

The following table shows the various filenames you can generate from an input filename (`%C`) of `/users/maxwell/circuits/opamp.ckt`:

Colon Modifier	Comments	Output Filename Result
`%C`	Input filename	`/users/maxwell/circuits/opamp.ckt`
`%C:r`	Root of the input filename	`/users/maxwell/circuits/opamp`
`%C:e`	Extension of the input filename	`ckt`
`%C:h`	Head of the input filename	`/users/maxwell/circuits`
`%C:t`	Tail of the input filename	`opamp.ckt`
`%C::`	Second colon is appended to the input filename and the end of the modification	`/users/maxwell/circuits/opamp.ckt:`
`%C:h:h`	The head of `%C:h` (such a recursive use of `:h` might be useful if you want to direct your output to a different directory from that of the input file)	`/users/maxwell`

Colon Modifier	Comments	Output Filename Result
`%C:t:r`	The root of `%C:t`	`opamp`
`%C:r:t`	The tail of `%C:r`	`opamp`
`/tmp%C:t:r.raw`	The suffix `.raw` is appended to the root of `%C:t`, and the full path is altered to put `opamp.raw` in the `/tmp` file.	`/tmp/opamp.raw`
`%C:e.%C:r:t`	Extension of `%C` followed by the tail of `%C:r`	`ckt.opamp`

Related Topics

Output Files of a Spectre FX Run

Using the Command-Line Option -outdir to Specify the Output Directory

Using the Command-Line Option -outname to Specify the Names of the Output Files

Reviewing the Log File

The Spectre FX log file provides important information about the simulation. Therefore, it is recommended to review the log file after starting the simulation.

The output log file displays the following information:

- Warning/Info/Error messages related to netlist parsing, and models
- Circuit inventory details
- Output statistics for probe/save/IC and measurement
- Simulation mode and important parameters/options
- Run time and memory consumption information
- Status of transient simulation
- Backannotation statistics for the post-layout simulation (for more information, refer to the EMIR Analysis and Post-Layout Simulations chapter)

The following topics discuss some of the important information stored in the log file and the options to control the log file in detail.

File Reading and Parsing Related Information

The Spectre FX log file provides information on the files read during a simulation run, as shown below.

```
Reading file: /projects/p1/simulation/Spectre/FX_run/top.sp
Reading file: /projects/p1/simulation/Spectre/FX_run/netlist.cdl
Reading file: /projects/p1/simulation/Spectre/FX_run/PDK/Spectre/vl.l/
16nm_model.scs
```

Spectre FX provides an option, print_section, with possible values of `no` and `yes`, that enables you to print the information about the library files read during the simulation. When this option is set to `no`, only the library file information is printed in the log file. When this option is set to `yes`, detailed sections of the library files read during the simulation are printed in the log file.

For example, if you specify `print_section=yes`, the log file displays the following information for the same example:

```
Reading file:  /projects/pl00/simulation/Spectre/FX_run/top.sp
```

```
Reading file:  /projects/p100/simulation/Spectre/FX_run/netlist.cdl
Loading section: SS from file: /projects/pl00/simulation/Spectre/FX_run//PDK/
Spectre/ v1.1/16nm model.scs
Loading section: TT BIP DIO from file: /projects/pl00/simulation/Spectre/FX_run//
PDK/Spectre/vl.1/16nm_model.scs
Loading section:TT RES DISRES from file: /projects/pl00/simulation/Spectre/FX_run/
/PDK/Spectre/v1.1/16nm_model.scs
```

Options from the Configuration File and Netlist

You can use the command-line option `+echooptions` to dump the options specified in the netlist and configuration files, as shown in the following example:

```
%> spectrefx -64 -f fsdb +echooptions +config acc_tuning.cfg -outdir
SpectreFX_results/ +mt=8 top_ba.sp
```

The options set in the netlist and configuration files are displayed in the log file. Spectre FX dumps all the valid options grouped by global and scoped (local) options.

```
Echo all options:
/case_path/top_ba.sp
        sfx_ba_dspf_net = test.dspf
        sfx_ba_dspf_instance = test.dspf
        sfx_ba_remove_prefix = m
/case_path/acc_tuning.cfg
        sfx_optimize = mx
        sfx_preset = ax subckt=vco

Global user options:
               temp = 30
sfx_ba_remove_prefix = m
       sfx_optimize = mx
    sfx_ba_dspf_net = test.dspf
sfx_ba_dspf_instance = test.dspf

Scoped user options:
         sfx_preset = ax       subckt=vco
```

Partition Information

The Spectre FX log file displays the information related to the biggest partition. It includes the number of nodes and elements including MOSFETs, resistors, and capacitors, as shown below.

```
Partition Information:
       The biggest SimUnit: nodes=101586 elements=471861 (M=30664 R=90902 C=350295)
```

Output and IC/Nodeset Related Information

The Spectre FX simulator provides useful information in the log file for the signals probed, measured, or set with IC/nodeset, especially with wildcards.

An example of the log file and the messages is shown below.

```
Output statements:
    .probe  4
   .measure 1
       save 0
Design checks inventory:
    dyn_dcpath 2
       dyn_exi 1
Wildcard match summary:
        probe v(*) depth= 5:     1723
        probe i(vv*) depth=1:    18
        ic ncored* 32768
        ic *.bl    254
        ic *.nbl   254
Output and IC/nodeset summary:
        probe  3288
        measure   1
        ic 6438
```

Transient Simulation Information

The status of the transient simulation is updated in the log file and screen output. You can use the sfx_progress_percentage option to control the frequency at which the information is updated in the log file.

Example

```
.option sfx_progress_percentage=10
```

The above statement states that the log file should be updated after every 10 percent of the transient simulation is completed.

The following is printed in the log file:

```
Opening the FSDB file out/sram16k_status.raw/../result1.fsdb ...
```

```
500.00 ns / 5.00 us (10.00%) # of steps: 13158 time spent: 00:00:07 time left:
00:01:10 CPU: 799.44% mem: 370.777 MB
  1.00 us / 5.00 us (20.05%) # of steps: 26394 time spent: 00:00:15 time left:
00:01:02 CPU: 798.56% mem: 373.973 MB
  1.51 us / 5.00 us (30.15%) # of steps: 39678 time spent: 00:00:23 time left:
00:00:55 CPU: 798.59% mem: 376.562 MB
  2.01 us / 5.00 us (40.10%) # of steps: 52738 time spent: 00:00:31 time left:
00:00:47 CPU: 798.59% mem: 376.562 MB
  2.50 us / 5.00 us (50.05%) # of steps: 65841 time spent: 00:00:39 time left:
00:00:39 CPU: 799.60% mem: 376.562 MB
  3.00 us / 5.00 us (60.03%) # of steps: 78965 time spent: 00:00:47 time left:
00:00:31 CPU: 799.62% mem: 376.562 MB
  3.50 us / 5.00 us (70.00%) # of steps: 92058 time spent: 00:00:55 time left:
00:00:23 CPU: 799.62% mem: 376.562 MB
  4.00 us / 5.00 us (80.02%) # of steps: 105246 time spent: 00:01:03 time left:
00:00:15 CPU: 799.63% mem: 376.562 MB
  4.50 us / 5.00 us (90.05%) # of steps: 118440 time spent: 00:01:11 time left:
00:00:07 CPU: 799.63% mem: 376.562 MB
Tran completed
Number of steps = 131507
```

The log file contains the following information related to the transient simulation:

- The waveform file opened at the beginning
- The current elapsed time
- The estimated time required to complete the rest of the transient simulation
- The current memory usage
- The number of steps taken to complete the transient simulation

If environment variable `SPECTRE_COMPATIBLE_FX` is set, the transient simulation progress format for Spectre FX is same as the format for Spectre, as shown below.

```
tran: time = 125 ns       (2.5 %), step = 8 ps         (160 u%)
tran: time = 375 ns       (7.5 %), step = 8 ps         (160 u%)
tran: time = 625 ns      (12.5 %), step = 1 ps          (20 u%)
tran: time = 875 ns      (17.5 %), step = 8 ps         (160 u%)
tran: time = 1.125 us    (22.5 %), step = 8 ps         (160 u%)
……
```

2

Netlist Compatibility and Support

The Spectre FX simulator shares the front-end of Spectre platform. It provides support for the native Spectre syntax and is fully compatible with HSPICE® (from Synopsys, Inc.) netlist files.

Spectre FX automatically sets SPICE or Spectre compatibility mode by detecting the input file format. You can specify `+spectre` or `+spice` at the command line to ensure the compatibility mode is as expected.

Related Topics

SPICE Control Statements Supported by Spectre FX

SPICE Options Supported by Spectre FX

SPICE Control Statements Supported by Spectre FX

Spectre FX supports the following SPICE control statements:

- .alter
- .connect
- .data
- .end
- .endl
- .ends or .eom
- .global
- .ic
- .if and .else
- .include
- .lib
- .malias
- .model
- .nodeset
- .op
- .options
- .param
- .subckt or .macro
- .temp
- .tran

Related Topics

.lprobe or .lprint

.probe or.print

.measure

.alter

```
.alter
```

Description

The `.alter` statement is used to repeat simulations under different conditions by modifying parameters, temperatures, models, circuit topology (different elements and subcircuit definitions), and analysis statements. You can use multiple `.alter` statements 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` statements 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 also treated in the same way.

Arguments

None

Examples

In the following example, the value of the capacitor in the first simulation run is `1pf`. In the second and third simulation runs, the values change to `100pf` and `10fF` respectively.

```
V0 (net5 0) vsource type=pulse val0=0.0 val1=1 period=40n rise=10p \
fall=10p width=20n
C0 (cnode 0) capacitor c=cValue
R0 (net5 cnode) resistor r=1K ....
.alter
.param Cvalue=1pF
.alter
.param Cvalue=100pF
.alter
.param Cvalue=10fF
.end
```

In the following example, the first simulation is run at `25 C` and the second simulation is run at `50 C`:

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

.connect

```
.connect node1 node2
```

Description

Connects node1 and node2.

Arguments

Name	Description
node1, *node2*	Node names.

Example

```
.connect vdd vdd!
```

Connects the `vdd` and `vdd!` nodes. If probed, the original nodes are retained in the waveform file.

.data

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

Description

Enables you to perform data-driven analysis in which parameter values can be modified in different simulations. This statement is used in conjunction with a .tran statement with the keyword `data=name`.

Arguments

Name	Description
name	Specifies the data name used in the analysis statements.
param1, *param2*	Specifies the parameter names used in the netlist file.
val11, *va21...*	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 Spectre FX simulator to perform two separate simulations with the two pairs of parameters, `res` and `cap`.

.end

```
.end
```

Description

Specifies the end of the netlist file description. All subsequent statements are ignored.

Arguments

None

Example

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

.endl

```
.endl
```

Description

Specifies the end of a library definition.

Arguments

None

Example

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

.ends or .eom

```
.ends
```

or

```
.eom
```

Description

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

Arguments

None

Example

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

or

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

.global

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

Description

Defines the global nodes. The Spectre FX simulator connects all the references to a global node name, at any hierarchical level, to the same node.

Arguments

Name	Description
node1 node2...	Node names.

Example

```
.global vdd gnd
```

Tells the Spectre FX simulator to define `vdd` and `gnd` as global nodes.

.ic

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

Description

Specifies the initial voltage condition for nodes. The Spectre FX 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 within 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 considered.

Arguments

Name	Description
`v(node)`	Sets the initial voltage for a node. The node name can be hierarchical and can contain wildcards (for example, `x?1.*.n*`). In this case, the Spectre FX 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 this parameter is equivalent to defining the statement within a subcircuit declaration.
`depth`	Specifies the depth in the circuit hierarchy to which a wildcard name applies. This parameter is available only when the `*` wildcard is used in the output variable. If set to `1`, only the nodes at the current level are applied (the default value is infinity).

Example

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

Tells the Spectre FX simulator to initialize node `n1` to `0.5 V` and node `n2` to `1.5 V` for all instances of the subcircuit `inv`.

.if and .else

```
.if (condition_1)
          Statements
     .elseif (condition_2)
          Statements
     .else
          Statements
     .endif
```

Description

Specifies the different conditions that determine the components that are instantiated by the Spectre FX simulator for a given simulation. The determining conditions are computed from the values of parameter expressions. You can specify these conditions with the structural `.if`/`.else` statement. These statements can be embedded.

Arguments

Name	Description
`condition`	Boolean-valued expressions where a non-zero value is considered as *true*.
`statements`	Contains one or more instance statements or `.if` statements. The `.else` or `.elseif` part of the statement is optional.

Example

The following example illustrates the use of the `.if` statement. There are additional `.if` statements in the *statement1* and *statement2* fields.

```
.param cvalue=1p
.if (rseries == 0)
c1 a b cvalue
.if (gparallel != 0)
rgp1 a b '1/gparallel'
.endif
.else
r2 a x rseries
c2 x b cvalue
.if (gparallel != 0)
rgp2 x b '1/gparallel'
```

```
.endif
.endif
```

In the above example, the Spectre FX simulator adds different instance statements into the simulation depending on the values of the parameters, `rseries` and `gparallel`.

- If both `rseries` and `gparallel` are zero, the simulator includes the instance statement for capacitor `c1`. If `rseries` is zero and `gparallel` is a non-zero value, the simulator includes the instance statements for capacitor `c1` and resistor `rgp1`.
- If `rseries` is non-zero and `gparallel` is zero, the simulator includes the instance statements for resistor `r2` and capacitor `c2`.
- If neither `rseries` nor `gparallel` is zero, the simulator includes the instance statements for resistor `r2`, capacitor `c2`, and resistor `rgp2`.

Related Topics

Conditional Instances

.include

`.include [filepath] filename`

Description

Inserts the contents of the specified file into the netlist file.

`[filepath] filename` can be enclosed within single or double quotation marks.

Arguments

Name	Description
filepath	(Optional) Path from where the specified file is to be read.
filename	Name of the file from which contents are to be inserted into the netlist.

Example

```
.include options.txt
```

Inserts the `options.txt` file into the netlist file.

.lib

`.lib [libpath] library_name section_name`

Description

Reads the common statements, such as device models, from a library file.

Arguments

Name	Description
`[libpath]`	Path to the library file.
`library_name`	Name of the library file.
`section_name`	Section of the library to be included.

Note: `[libpath]` and `library_name` can be enclosed within single or double quotation marks.

Example

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

Tells the Spectre FX simulator to read the `tt` section from the `models.lib` library file.

.malias

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

Description

Creates an alias name for a model. You can use alias names in the same way as *model_name*.

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

Arguments

Name	Description
alias_name	Specifies the alias name to be used for the model.
model_name	Specifies the model name.

.model

```
.model name type [param1=value1 ... [param2=value2 ]]
```

Description

Allows certain parameters, which are expected to be shared over many instances, to be specified once. However, for any given component, it is predetermined which parameters can be specified using the `.model` statement for that component.

Arguments

Name	Description
name	Name of the model.
type	Specifies the type name of the component. Typical device types are: ■ `C`: Capacitor Model ■ `D`: Diode Model ■ `NMOS`: N-channel MOSFET Model ■ `NPN`: npn Bipolar Junction Transistor (BJT) Model ■ `PMOS`: P-channel MOSFET Model ■ `PNP`: pnp Bipolar Junction Transistor (BJT) Model ■ `R`: Resistor Model
parameter=value	This is optional. You can repeat it any number of times in a `.model` statement. Each parameter specification is a model parameter, specified as `parameter=`*`value`*. For more information about the available model parameters for each component, refer to the *Spectre Circuit Simulator Components and Device Models Reference manual*.

Example

```
.model nch nmos level = 72 version = 105.03 bulkmod=0 geomod=2
```

Here, a NMOS model with `level=72` is defined. It is a BSIM-CMG model.

.nodeset

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

Description

Specifies the nodesets for the node. The Spectre FX 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 within the scope of a subcircuit. In this case, the initial guess is assigned to the nodes local to the subcircuit. If a conflict occurs between an embedded nodeset and a hierarchical nodeset, the embedded one is considered.

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

Arguments

Name	Description
`v(`*node*`)`	Specifies the nodeset for the node. The node name can be hierarchical and can contain wildcards (for example, `x?1.*.n*`). In this case, the Spectre FX simulator assigns the initial condition to all the nodes that match the name.

Example

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

In the above example, 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 because `.nodeset` is used only at the first iteration.

.op

```
.op  [time1] [time2]...[timeN]
```

Description

Performs an operating point analysis. The Spectre FX simulator reports all the node voltages in a `.ic0` file. If multiple time points are specified, the Spectre FX simulator saves the node voltages at all the given time points. If the time is not specified, the Spectre FX simulator performs the operating point analysis at time 0. If the time point in `.op` is greater than the stop time of the `.tran` analysis, the Spectre FX simulator does not perform the operating point analysis.

Note: Only the voltage format is supported. You cannot specify any other format.

Arguments

Name	Description
time	Specifies the time at which the report is printed. The unit is seconds. The default value is `0 s`.

Example

```
.op 0 35n 200n
```

Tells the Spectre FX simulator to calculate the operating point at `0 s`, `35 ns`, and `200 ns`. The simulator generates the output files, as shown below.

*example*_op.ic0
*example*_op.ic0@3.5e-08
*example*_op.ic0@2e-07

.options

`.options` *option1* [*option2*] ...

Description

Specifies a list of SPICE options to be used in the Spectre FX simulation.

Arguments

Name	Description
list of SPICE options	A space-separated list of SPICE options.

Related Topics

SPICE Options Supported by Spectre FX

.param

```
.param param1=value1 [param2 = value2 ... paramn = valuen]
```

or

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

Description

Defines the 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'
```

.pat

```
Vxx n1 n2 PAT (vhigh vlow td tr tf tsample pat_name [RB=val] [R=val])
Ixx n1 n2 PAT (ihigh ilow td tr tf tsample pat_name [RB=val] [R=val])
.PAT pat_name=data [RB=val] [R=val]
```

Description

Defines a time-varying pattern that can be used in independent voltage or current sources. The patterns are used to specify digital bit strings. The parameter data, which is a string that consists of characters `1`, `0`, `m`, and `z`, must always be specified. For example, `data = "1011mz11"`. The patterns can be embedded within another pattern.

Arguments

Name	Description
vhigh	High voltage for pattern voltage sources (unit: Volts).
ihigh	High current for pattern current sources (unit: Amps).
vlow	Low voltage for pattern voltage sources (unit: Volts).
ilow	Low current for pattern current sources (unit: Amps).
td	Delay time of pattern sources (unit: seconds).
tr	Time of rising edge from low value to high value (unit: seconds).
tf	Time of falling edge from high value to low value (unit: seconds).
tsample	Time spent in one bit of pattern. For high value, it contains the `tr` time. For low value, it contains the `tf` time. (unit: seconds).
pat_name	The pattern defined in the `.pat` statement. The following four values are supported in the string: ■ `1`: High voltage for voltage sources or high current for current sources. ■ `0`: Low voltage for voltage sources or low current for current sources. ■ `m`: the middle voltage, `(vhigh+vlow)/2`, or the middle current, `(ihigh+ilow)/2`. ■ `z`: HighZ state for the pattern sources.

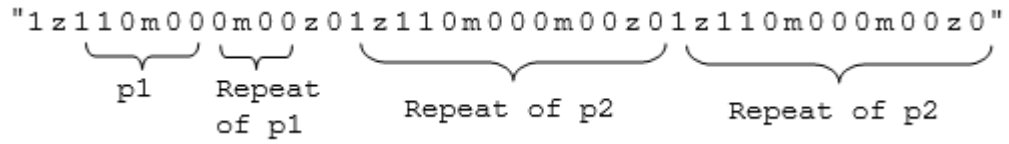
Name	Description
RB=val	The position of the bit that is used as the starting bit while repeating the string. The string is repeated from the bit specified using this parameter till the last bit present in the string. The parameter should be an integer greater than 1. It can go up to the length of the bit string. The default value is 1.
R=val	The number of times the string is to be repeated. The output maintains the state of the last bit after the last repeat. The parameter should be a non-negative integer. The default value is 0.

Examples

In the following example, pattern `p1` is embedded within another pattern `p2`:

```
vin in gnd PAT ('vccr' 0 1n 2n 5u 10u p2)
.pat p1=b110m00 rb=3 r=1
.pat p2=[b1z p1 z0] rb=1 r=2
```

The data of `p2` appears as follows:



In the following example, the pattern of voltage source `vin_2` is the same as `vin` in the first example:

```
vin_2 in_2 gnd PAT ('vccr' 0 1n 2n 5u 10u [b1z p1 z0] rb=1 r=2)
.pat p1=b110m00 rb=3 r=1
```

.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

The `.subckt` or `.macro` statements specify the beginning of a subcircuit definition. A subcircuit can have zero ports when all the nodes used in the subcircuit definition are declared as global. A subcircuit definition can contain elements, subcircuit calls, nested subcircuit definitions, as well as simulation output statements. The parameters can be declared within the subcircuit definitions using the `.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
```

Tells the Spectre FX simulator to define 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 the values for `w` and `l`.

.temp

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

Description

Defines the values of the temperature used in simulations.

Arguments

None

Example

```
.temp 0 50 100
```

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

.tran

```
.tran tstep tstop [uic][start = tstart]
or
.tran tstep tstop [uic][start = tstart] + [sweep swp_type]
or
.tran tstep tstop [uic][start = tstart] + [sweep data="data_name"]
```

Description

Defines the transient analysis in which the DC operating point is calculated first 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 simulator sets the node voltages as defined by the `.ic` statements (or by the `ic = parameters` in various element statements) and sets the unspecified nodes to 0 volts instead of solving the quiescent operating point. The DC operating points of the unspecified nodes are set to 0 volts.

Arguments

Name	Description
tstep	Since the `.print` statement is not supported in the original way and is converted to `.probe`, the `tstep` value is ignored in the Spectre FX simulation.
tstop	Specifies the stop time of the transient simulation.
uic	Ignores the DC operating point calculation and sets the unspecified nodes to 0 volts.
start	Specifies the start time to write the output waveforms. The transient simulation still starts from `t=0`; however, no waveform is saved before the start time.
sweep	Defines the sweep analyses in the transient simulation. The following three types of sweeps are supported: ■ Parameter sweep with `DEC`, `LIN`, `OCT`, or `PCI` ■ Data-driven sweep with the .data statement ■ Monte Carlo sweep

Name	Description
tempvec=(t1 temp1 t2 temp2)	Specifies the time points and values pair of temperature change during transient simulation.
tempstep	Specifies whether the time points and values pair given by the tempvec parameter is to be updated in one step, or as a series of steps.

Examples

The following example performs the transient analysis from 0 ns to 10 ns. In addition, *vcc* is swept linearly for five values from 2.0 to 3.0:

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

The following example performs the transient analysis from 0 ns to 1 us. In addition, the dataset allpars is used for performing the sweep:

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

The following example performs the transient analysis from 0 ns to 200 ns in steps of 1 ns without calculating the DC operating point since uic is used:

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

The following example performs the transient analysis from 0 ns to 200 ns. In addition, the temperature increases by 5C every 10ns from 0ns to 50ns:

```
.tran 1ns 200ns tempvec=(0 25 50n 50) tempstep=10n
```

SPICE Options Supported by Spectre FX

Spectre FX supports the following SPICE options that you can use with the `.options` or `.option` SPICE command:

- autostop
- geoshrink
- gmin
- measdgt
- parhier
- scale
- scalm
- search
- tnom

Related Topics

.options

autostop

```
.option autostop
```

Description

Stops the transient simulation when all the measurements are successfully completed. By default, `autostop` is disabled which means that the full transient simulation is run.

Example

When the `autostop` option is specified, Spectre FX stops the transient simulation and the following message is displayed in the log file:

```
Auto stop simulation at 1.79979e-06s
Tran completed
Number of steps = 6746
```

geoshrink

```
.option geoshrink=value
```

Description

The `geoshrink` option is an alias of the Spectre option `scalefactor`. It is used for Device Model Technology Scaling. The `options` parameter `scalefactor` enables device model providers to scale the device technology independent of the design dimension scaling done by circuit designers. The resulting device instance scaling is defined by `'scale * geoshrink'`.

Unlike the `options` parameter `scale`, `scalefactor` cannot be used as a netlist parameter and cannot be altered or used in `sweep` statements.

Example

```
.option geoshrink=0.9
.option scale=1e-6
```

The size of the devices is shrunk by `0.9`. The compounded scaling of the device instance dimension is `0.9e-6` since `scale` is specified as `1e-6`.

gmin

```
.option gmin=value
```

Description

Specifies the minimum conductance allowed for transient analysis. The default value is `1e-12`.

measdgt

```
.option measdgt=value
```

Description

Sets the number of significant digits for the results in the `.mt0` and `.measure` files. The default value is `5`.

parhier

```
.option parhier=local | global
```

Description

Specifies the rules for parameter passing; applies only to parameters with the same name, but under different levels of hierarchy.

- `local`– Tells the Spectre FX simulator that a parameter name in a subcircuit overrides the same parameter name at a higher level in the hierarchy.
- `global`– Tells the Spectre FX simulator that a parameter name at a higher level of the hierarchy overrides the same parameter name at a lower level.

scale

```
.option scale=value
```

Description

Specifies the scaling factor used to scale the parameters in the element card. The default value is `1`.

scalm

```
.option scalm=value
```

Description

Models the scaling factor used to scale the model parameters defined in model cards. The default value is `1`.

search

```
.option search="path"
```

Description

Specifies the search path for libraries and included files. While reading the netlist, Spectre FX searches the files in the specified path. It is similar to the `-I <path>` Spectre command-line option.

tnom

```
.option tnom=value
```

Description

Specifies the reference temperature for the simulation. The default value is 27 degrees Celsius in the Spectre compatible mode, and 25 degrees Celsius in SPICE compatible mode.

3

Circuit Components, Models, Stimuli, and Verilog-A Support

Spectre FX shares the same infrastructure as other tools in Spectre for device models, built-in components, digital vector, VCD/EVCD, and Verilog-A support. Their definitions are supported in both Spectre and SPICE syntax.

Related Topics

Support for Circuit Components and Device Models

Support for Digital Vector File

Support for Verilog Value Change Dump Stimuli

Support for Verilog-A Models

Support for Circuit Components and Device Models

Spectre FX supports the same circuit components and device models that are supported by other simulators in the Spectre platform.

Related Topics

Spectre Circuit Simulator Components and Device Models Reference

Support for Digital Vector File

A digital vector file (VEC) defines how to perform vector checks and apply stimuli according to the digital vectors. To process the digital vector file formats, specify the following statement in the netlist file:

Spectre Syntax

```
vec_include "vector_filename" [HLCheck=0|1] [autostop=yes|no] [insensitive=yes|no]
```

SPICE Syntax

```
.vec "vector_filename" [HLCheck=0|1] [autostop=true|false] [insensitive=yes|no]
```

Arguments

Name	Description
vector_filename	The filename of the digital vector file.
HLCheck = 0 | 1	Special flag that enables the checking for the H and L states for input signals. The default value is 0.
autostop=yes|no	If the value is set to no, Spectre FX uses the end time from the .tran or tran statement. This is the default value. If the value is set to yes, Spectre FX uses the last specified time point in the vector file as the end time. If multiple .vec files are specified, and autostop=yes is specified in one or all .vec statements, the simulator takes the largest time point available in the .vec files and uses it as the end time. **Note:** The autostop argument can also be used when loading the .vcd and .evcd files.
insensitive=yes|no	Specifies whether the vector file content is considered case sensitive or insensitive. The default value is yes. For example, if you are in Spectre mode and a vector file is case sensitive, use insensitive=no.

Related Topics

Digital Vector File Format

Support for Verilog Value Change Dump Stimuli

The Spectre FX circuit simulator supports the Verilog® value change dump (VCD) and extended VCD (EVCD) file formats. It requires a signal information file to define the signal characteristics.

The VCD file (ASCII format) contains information about the value changes for selected variables in the circuit design. Spectre FX supports the following two types of VCD files:

- Four-state in VCD – represents variable changes in `0`, `1`, `x` (unknown or "not needed") and `z` (tri-state) without providing strength information and port direction.
- Extended in EVCD – represents variable changes in all the states and provides the strength information and port direction.

To process the VCD or EVCD files in Spectre FX, specify one of the following statement the netlist file:

Spectre Syntax

```
vcd_include "vcd_filename" "signal_info_filename" [autostop=yes|no]
     [insensitive=yes|no]
```

or

```
evcd_include "evcd_filename" "signal_info_filename" [autostop=yes|no]
     [insensitive=yes|no]
```

SPICE Syntax

```
.vcd "vcd_filename" "signal_info_filename" [autostop=yes|no] [insensitive=yes|no]
```

or

```
.evcd "evcd_filename" "signal_info_filename" [autostop=yes|no]
     [insensitive=yes|no]
```

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, Spectre FX can share the same signal information file.

By default, Spectre FX handles the VCD and EVCD file content as case-insensitive (`insensitive=yes`). Case sensitivity can be set using the `insensitive=no` option.

Related Topics

Verilog Value Change Dump Stimuli

Support for Verilog-A Models

The Cadence® Verilog®-A language is the analog subset of the Verilog-AMS language. With Verilog-A, you can create and use modules that describe the high-level behavior of components and systems. The Verilog-A language is a high-level language that uses modules to describe the structure and behavior of analog systems and their components. With the analog statements of Verilog-A, you can describe a wide range of conservative systems and signal-flow systems, such as electrical, mechanical, fluid dynamic, and thermodynamic systems. To describe a system, you must specify both the structure of the system and the behavior of its components.

The following is the syntax to invoke a Verilog-A model:

Spectre Syntax

```
ahdl_include "[filepath]veriloga_file.va"
```

SPICE Syntax

```
.hdl [filepath]veriloga_file.va
```

Related Topics

Cadence Verilog-A Language Reference

4

Analyses Supported in Spectre FX

Spectre FX supports the following analyses:

DC Analysis

AC Analysis

Transient Analysis

Monte Carlo Analysis

Reliability Analysis

DC Analysis

The DC analysis finds the DC operating point or DC transfer curves of the circuit. To generate transfer curves, specify a parameter and a sweep range. The swept parameter can be circuit temperature, a device instance parameter, a netlist parameter, or a subcircuit parameter for a particular subcircuit instance.

You can sweep the circuit temperature by giving the parameter name as `param=temp` with no `dev` or `sub` parameter. You can sweep a top-level netlist parameter by giving the parameter name with no `dev` or `sub` parameter.

You can sweep a subcircuit parameter for a particular subcircuit instance by specifying the subcircuit instance name with the `sub` parameter and the subcircuit parameter name with the `param` parameter. After the analysis is complete, the modified parameter returns to its original value.

Syntax for DC Analysis

```
Name dc parameter=value ...
```

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (step, lin, log, dec) and determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. If you specify the oppoint parameter, Spectre FX computes and outputs the linearized model for each nonlinear component.

Nodesets help find the DC or initial transient solution. You can supply them in the circuit description file with `nodeset` statements, or in a separate file using the `readns` parameter. When nodesets are given, Spectre FX computes an initial guess of the solution by performing a DC analysis while forcing the specified values onto nodes by using a voltage source in series with a resistor whose resistance is `rforce`. Spectre FX then removes these voltage sources and resistors and computes the true solution from this initial guess.

Nodesets have two important uses:

- If a circuit has two or more solutions, nodesets can bias the simulator towards computing the desired one.
- They are a convergence aid. By estimating the solution of the largest possible number of nodes, you might be able to eliminate a convergence problem or dramatically speed convergence.

When you simulate the same circuit many times, we suggest that you use both the `write` and `readns` parameters and give the same filename to both parameters. The DC analysis then converges quickly even if the circuit has changed somewhat since the last simulation, and the nodeset file is automatically updated.

AC Analysis

The AC analysis linearizes the circuit about the DC operating point and computes the response to all specified small sinusoidal stimulus.

The Spectre simulator can perform the analysis while sweeping a parameter. The parameter can be frequency, temperature, component instance parameter, component model parameter, or netlist parameter. If changing a parameter affects the DC operating point, the operating point is recomputed on each step. You can sweep the circuit temperature by giving the parameter name as `temp` with no `dev` or `mod` parameter. You can sweep a netlist parameter by giving the parameter name with no dev or mod parameter. After the analysis has completed, the modified parameter returns to its original value.

Syntax for AC Analysis

```
Name ac parameter=value ...
```

You can specify sweep limits by giving the end points or by providing the center value and the span of the sweep. Steps can be linear or logarithmic, and you can specify the number of steps or the size of each step. You can give a step size parameter (`step,lin,log,dec`) to determine whether the sweep is linear or logarithmic. If you do not give a step size parameter, the sweep is linear when the ratio of stop to start values is less than 10, and logarithmic when this ratio is 10 or greater. All frequencies are in Hertz.

The small-signal analysis begins by linearizing the circuit about an operating point. By default, this analysis computes the operating point if it is not known or recomputes it if any significant component or circuit parameter has changed. However, if a previous analysis computed an operating point, you can set `prevoppoint=yes` to avoid recomputing it. For example, if you use this option when the previous analysis was a transient analysis, the operating point is the state of the circuit on the final time point.

AC Analysis Parameters Supported by Spectre FX

Sweep Interval Parameters

Parameter	Description
start	Start sweep limit
stop	Stop sweep limit
center	Center of sweep

span	Sweep limit span
step	Step size, linear sweep
lin	Number of steps, linear sweep
dec	Points per decade
log	Number of steps, log sweep
values	Array of sweep values
valuesfile	Name of the file containing the sweep values

Sweep Variable Parameters

Parameter	Description
mod	Model whose parameter value is to be swept
param	Name of parameter to sweep
freq	Frequency when a parameter other than frequency is being swept

State-file Parameters

Parameter	Description
readns	File that contains an estimate of the DC solution
write	DC operating point output file at the first step of the sweep

Initial condition Parameters

Parameter	Description
force	The set of initial conditions to use. Possible values are none, node, dev, and all.
readforce	File that contains initial conditions

Output Parameters

Parameter	Description
save	Signals to output. Possible values are all, lvl, allpub, lvlpub, selected, none, and no-output

Examples

The following examples show how you can use the various parameters to run AC analysis:

```
ac1 ac start=0 stop=10G step=2G
;; Runs an AC analysis ac1 at a linear sweep of frequencies from 0 to 10G with a
step at each 2G
ac1 ac center=30M span=1M
;; Runs an AC analysis ac1 at swept frequencies spanning over 1M with center at 30M


ac1 ac start=0 stop=10G lin=100 //Number of steps = 101
; Runs an AC analysis ac1 at 100 steps starting from 0 to 10G. It runs a total of
101 steps
ac1 ac start=1 stop=10G dec=10
ac1 ac start=1 stop=10G log=50
ac1 ac values=[1M 100M 1G 5G 7G 9G 9.5G 10G]
ac1 ac valuesfile="ac_valuesfile.txt"
ac1 ac dev=v_pulse2 param=dc start=-0.1 stop=5.1 step=0.1 freq=1k
ac2 ac start=7.5e4 stop=8e4 step=1e3 mod=nch freq=1M param=vsat
ac2 ac freq=6G  param=Cap1 start=200f stop=300f lin=10 annotate=status
ac_all ac start=1K stop=10G dec=10 force=all write="force_all.ac"
ac_none ac start=1K stop=10G dec=10 force=none  write="force_none.ac"
ac_node ac start=1K stop=10G dec=10 force=node write="force_node.ac"
ac_dev ac start=1K stop=10G dec=10 force=dev  write="force_dev.ac"
ac_readforce ac start=1K stop=10G dec=10 readforce="./force.ic" force=all
ac1 ac start=1 stop=10G dec=10 save Rind1
```

Transient Analysis

Spectre FX supports only a few parameters of the `tran` statement in Spectre. The supported parameters are used to save or recover state and output starting point. The parameters related to the solver are not supported.

Spectre FX supports the following transient analysis parameters:

- Simulation Interval Parameters for tran
- Initial Condition Parameters for tran
- Convergence Parameter for tran
- State-File Parameters for tran
- Strobe Output Parameters for tran
- Dynamic Parameters for tran

Note: All examples shown below are in Spectre syntax.

Simulation Interval Parameters for tran

`stop (s)`	Specifies the stop time for the transient analysis. The `stop` parameter must be specified in the `tran` statement. **Syntax** `tran1 tran stop=`*`tstop`*
`start=0 s`	Specifies the start time when the waveform is generated for the transient analysis. Although the simulation starts from 0s, the waveform is not generated before the specified start time. This parameter is the same as the `outputstart` parameter. **Syntax** `Tran1 tran stop=` *`tstop`* `start=`*`tstart`*
`outputstart=start s`	Specifies that the output be saved only after the time specified with the `outputstart` parameter is reached. The default value of `outputstart` is inherited from the parameter `start`. **Syntax** `Tran1 tran stop=`*`tstop`* `outputstart=`*`toutputstart`*

Initial Condition Parameters for tran

`skipdc=no`	If set to yes, there is no DC analysis for transient. In Spectre FX, only the values no and yes are supported. The default value is `no`. **Syntax** `Tran1 tran stop=`*`tstop`* `skipdc=no | yes` **Example** `Tran1 tran stop=1m skipdc= yes` In the Spectre FX log file, the following message is displayed indicating that DC is skipped before the transient simulation. `Starting Skipping DC Simulation ...` `Time for Skipping DC Simulation: CPU = 6.999 ms, elapsed = 452.995 us.`
`readic`	Specifies the file that contains the initial condition (IC) information. The format of the IC file in Spectre syntax is different from that in SPICE syntax. The file specified in the SPICE syntax is invoked using the `.include` statement. The IC file for the parameters `readic` and `readns` is generated using the `write` parameter of the Spectre `tran` statement. **Syntax** `tran1 tran stop=`*`tstop`* `readic=`*`"filename.ic"`*

Convergence Parameter for tran

`readns`	Specifies the file that contains an estimate of the initial transient solution. For IC file reading using `readic`, the nodes voltages strictly follow the specified values at the beginning of the transient simulation; however, if you use `readns`, it just provides an initial guess for the operating point calculation. The final operating points can be different from the specified ones. **Syntax** `Tran1 tran stop=`*`tstop`* `readns=`*`"filename.ic"`*

State-File Parameters for tran

`write`

Specifies the file to which the initial transient solution needs to be written at `t=0s` of the transient analysis.

Syntax

```
Tran1 tran stop=tstop write="filename.ic"
```

`savetime=[...]`

Saves the analysis states to files at the specified time points.

Syntax

```
Tran1 tran stop=tstop savetime=[point1 point2 …
     pointN]
```

Example

The top level netlist `save_state.scs` contains the following:

```
tran1 tran stop=10m savetime=[1m 1.5m 2.03m]
```

In the directory of output files, the state files are saved as follows:

```
save_state.scs.save_state.srf_at_1.00ms
save_state.scs.save_state.srf_at_1.50ms
save_state.scs.save_state.srf_at_2.03ms
```

`savefile`

Specifies the name of the `savestate` files.

Syntax

```
tran-name tran stop=tstop savefile="filename"
```

Example

The top level netlist `save_state.scs` contains the following:

```
tran1 tran stop=10m savetime=[1m 1.5m 2.03m]
savefile="ss_file"
```

In the directory of output files, the state files are saved as follows:

```
ss_file_at_1.00ms
ss_file_at_1.50ms
ss_file_at_2.03ms
```

`recover`

Specifies the file to be restored. The transient simulation does not start from `t=0s`, but from the time point where the `statefile` was saved. The recover file can also be defined using the command-line option `+recover`.

Syntax

tran-name tran stop=*tstop* recover=*"filename"*

Example

```
tran1 tran stop=10m recover="ss_file_at_1.00ms"
```

The Spectre FX log file indicates the transient simulation is recovered and resumed at `1ms`, as shown below.

```
Transient Analysis `recover': time = (1 ms -> 10 ms)
************************************************
Important parameter values:
    start = 1 ms
    outputstart = 1 ms
    stop = 10 ms
    temp = 25 C
    tnom = 25 C

Recovering from save-restart file ss_file_at_1.00ms,
Restarting at time 1 ms.
```

Strobe Output Parameters for tran

`skipstart=0 s`	The time to start skipping output data. Example: `tran1 tran stop=10u strobestart=1u strobeperiod=1/ 1e9/64` The value of strobeperiod is the expression '1/1e9/ 64'=15.625ps and the strobe output starts from 1us.
`skipstop=stop s`	The time to stop skipping output data.
`strobeperiod=0 s`	The output strobe interval (in seconds) of transient time.
`strobedelay=0 s`	The delay (phase shift) between the skipstart time and the first strobe point.
`strobeoutput=strobeonly`	Specifies which time points to output during strobe. Possible values are `strobeonly`, `all`, and `none`.
`strobestep=0 s`	Equivalent to strobeperiod.
`strobefreq`	The reciprocal of strobeperiod (strobestep).
`strobestart=0 s`	Equivalent to skipstart.
`strobestop=stop s`	Equivalent to skipstop.
`strobetimes=[...] s`	Times in ascending order when strobe output was performed.

Dynamic Parameters for tran

`param=`*`param_name`*	Dynamic parameter name. The parameter name can be a design parameter, maxstep, speed or temp. **Syntax** *`tran-name tran stop=tstop param= param_name param_vec=[ t1 val1 t2 val2...]`* **Example** `tran1 tran stop=10u param=temp param_vec=[0n 20 50n 25 100n 75]` `tran1 tran stop=10u param=speed param_vec=[0n lx 50n mx]`

`param_vec=[` *`t1 val1 t2 val2...`*`]`	Time points and values of the parameter.
`param_file=`*`file`*	File name if the `param_vec` is defined in a separate file.
`paramset`	Name of the dynamic parameter set.

Monte Carlo Analysis

The montecarlo analysis is a swept analysis with associated child analyses similar to the sweep analysis, see `spectre -h sweep`. The Monte Carlo analysis refers to statistics blocks, where statistical distributions and correlations of netlist parameters are specified.

For each iteration of the Monte Carlo analysis, new pseudorandom values are generated for the specified netlist parameters (according to their specified distributions) and the list of child analyses are then executed.

The Monte Carlo option allows for scalar measurements to be linked with the Monte Carlo analysis. Calculator expressions are specified that can be used to measure circuit output or performance values (such as the slew rate of an operational amplifier). During a Monte Carlo analysis, these measurement statement results vary as the netlist parameters vary for each Monte Carlo iteration and are stored in a scalar data file for post processing. By varying the netlist parameters and evaluating these measurement statements, the Monte Carlo analysis becomes a tool that allows you to examine and predict circuit performance variations that affect yield.

The statistics blocks allow you to specify batch-to-batch (process) and per- instance (mismatch) variations for netlist parameters. These statistically varying netlist parameters can be referenced by models or instances in the main netlist and can represent IC manufacturing process variation or component variations for board-level designs. The following description gives a simplified example of the Monte Carlo analysis flow:

```
perform nominal run if requested
if any errors in nominal run then stop
for each Monte Carlo iteration {
   if process variations specified then
      apply "process" variation to parameters
   if mismatch variations specified then
      for each subcircuit instance {
          apply "mismatch" variation to parameters
      }
   for each child analysis {
      run child analysis
         evaluate any export statements and
         store results in a scalar data file
      }
}
```

The following is the syntax for the Monte Carlo analysis:

```
Name montecarlo parameter=value … {
   analysis statements …
   export statements …
}
```

The Monte Carlo analysis:

- Refers to the statistics blocks for how and which netlist parameters to vary
- Generates statistical variation (random numbers according to the specified distributions)
- Runs the specified child analyses (similar to the Spectre FX nested sweep analysis), where the child analyses are either
  - Multiple data-producing child analyses (such as DC or AC analyses)
  - A single sweep child analysis, which itself has child analyses
- Calculates the export quantities

  Each Monte Carlo run processes `export` statements that implicitly refer to the result of the child analyses. These statements calculate scalar circuit output values for performance characteristics, such as slew rate.
- Organizes the export data appropriately

  Scalar data, such as bandwidth or slew rate, is calculated from an `export` statement and saved to an ASCII file, which can be used later for plotting a histogram or scattergram.
- After the Monte Carlo analysis is complete, all parameters are returned to their original values.

Related Topics

Monte Carlo Parameters Supported by Spectre FX

Specifying the First Iteration Number

Specifying Parameter Distributions Using Statistics Blocks

Multiple Statistics Blocks

Specifying Distributions

Distributed Monte Carlo Analysis

Monte Carlo Parameters Supported by Spectre FX

Monte Carlo Analysis Parameters

`numruns=100`	Number of Monte Carlo iterations to perform (not including nominal).
`firstrun=1`	Starting iteration number.
`runpoints=[...]`	Specifies the iteration indices to be simulated. Two types of settings are accepted; integers and ranges. For example, runpoints=[10 range(15, 18) 20] is an acceptable setting. It specifies that simulation needs to be performed for the 10th, 15th, 16th, 17th, 18th, and 20th iterations.
`variations=process`	Level of statistical variation to apply. Possible values are `process`, `mismatch`, or `all`.
`sampling=standard`	Method of statistical sampling to apply. Possible values are `standard`, `lhs`, and `lds`.
`seed`	Optional starting seed for random number generator.
`scalarfile`	Output file that will contain output scalar data.
`paramfile`	Output file that will contain output scalar data labels.
`dut=[...]`	If set, then the specified subcircuit or device instances will have process and mismatch variations applied, the unspecified instances will only have process variations applied. All subcircuits or devices instantiated under this instance will also have process and mismatch variations enabled. By default, mismatch variation is applied to all subcircuit or device instances in the design and process is applied globally. This parameter allows the testbench to change and not affect the variations seen by the actual design.
`ignore=[...]`	If set, no variation is applied to the specified subcircuit or device instance(s). All subcircuits or devices instantiated under this instance will also have no variation enabled. By default, mismatch is applied to all subcircuit or device instances in the design and process is applied globally.

Saving Process Parameters

`saveprocessparams`	Whether or not to save scalar data for statistically varying process parameters which are subject to process variation. Possible values are `no` or `yes`.
`processscalarfile`	Output file that will contain process parameter scalar data.
`processparamfile`	Output file that will contain process parameter scalar data labels.

Saving Mismatch Parameters

`dumpdependency`	Whether or not to save the dependency map. Possible values are `none` and `mismatch`.
`dependencyscalarfile`	Output random numbers to a file that are used by mismatch parameters.
`dependencyparamfile`	Output the mapping from the mismatch parameters to the corresponding subcircuit instances to a file.

Flags

`donominal=yes`	Whether or not to perform nominal run. Possible values are no or yes.
`nullmfactorcorrelation =no`	Controls the mismatch variation correlation of parallel devices defined by m-factor. If set to yes, devices are assumed to get uncorrelated mismatch variations. If set to no, devices are assumed to get the same mismatch variation. Possible values are no and yes.
`savefamilyplots=no`	Whether or not to save data for family plots. If yes, this could require a lot of disk space. Possible values are no or yes.
`savedatainseparatedir= no`	Whether or not to save data for an each plot in a separate directory. Possible values are no or yes. **Note:** Setting this parameter to yes may require a lot of disk space.

`evaluationmode=no`	If set to yes, dumps random numbers used in montecarlo analysis without running any enclosed analyses. Possible values are no and yes.
`wfseparation=no`	If set to yes, a separate directory for nominal run is created by the name nom and the directory names of individual iterations are simplified by removing the leading analysis names. Possible values are no and yes.
`usesamesequence=no`	If set to yes, the random sequence is maintained for a seed even if there is an empty dut/ignore list. Possible values are no or yes.

Annotation Parameters

`ignoremode`	Controls the type of variation that is applied to ignored devices when variations = `all`. By default, the variation seen by ignored devices depends on whether the dut or ignore parameter is used. If dut is used, then all non-DUT devices will get process variation. If the ignore parameter is used, then devices in the ignore list will use nominal values, that means the devices have no variation. When `ignoremode` = `nominal` | `process`, the behavior is consistent for both dut and ignore. If `ignoremode` = `nominal`, then all devices in the ignore list or devices that are not in the dut list will use nominal values. If `ignoremode` = `process`, all devices in the ignore list or devices that are not in the dut list have process variation. Possible values are `default`, `nominal`, `process` and `both`.

Specifying the First Iteration Number

The advantages of using the `firstrun` parameter to specify the first iteration number are as follows:

- You can reproduce a particular run from a previous experiment when you know the starting seed and run number but not the corresponding seed.
- If you are a standalone Spectre FX user, you can run a Monte Carlo analysis of 100 runs, analyze the results, decide they are acceptable, and then decide to do a second analysis of 100 runs to give a total of 200 runs. By specifying the `firstrun=101` for the second analysis, the Spectre FX simulator retains the data for the first 100 runs and runs only the second 100 runs. This gives the same results and random sequence as if you ran just a single Monte Carlo analysis of 200 runs.

Specifying Parameter Distributions Using Statistics Blocks

The statistics blocks are used to specify the input statistical variations for a Monte Carlo analysis. A statistics block can contain one or more process blocks (which represent batch-to-batch type variations) and/or one or more mismatch blocks (which represent on-chip or device mismatch variations), in which the distributions for parameters are specified. Statistics blocks can also contain one or more correlation statements to specify the correlations between specified process parameters and/or to specify correlated device instances (such as matched pairs). Statistics blocks can also contain a `truncate` statement that can be used for generating truncated distributions.

The statistics block contains the distributions for parameters:

- Distributions specified in the process block are sampled once per Monte Carlo run, are applied at global scope, and are used typically to represent batch-to-batch (process) variations.
- Distributions specified in the mismatch block are applied on a per-subcircuit instance basis, are sampled once per subcircuit instance, and are used typically to represent device-to-device (on chip) mismatch for devices on the same chip.

When the same parameter is subject to both process and mismatch variations, the sampled process value becomes the mean for the mismatch random number generator for that particular parameter.

Note: Statistics blocks can be specified using combinations of the Spectre FX keywords `statistics`, `process`, `mismatch`, `vary`, `truncate`, and `correlate`. Braces ({}) are used to delimit blocks.

The following example shows some sample statistics blocks, which are discussed after the example along with syntax requirements.

The following example shows some sample statistics blocks, which are discussed after the example along with syntax requirements.

```
// define some netlist parameters to represent process parameters
// such as sheet resistance and mismatch factors
parameters rshsp=200 rshpi=5k rshpi_std=0.4K xisn=1 xisp=1 xxx=20000 uuu=200

// define statistical variations, to be used
// with a MonteCarlo analysis.
statistics {
    process {   // process: generate random number once per MC run
        vary rshsp dist=gauss std=12 percent=yes
        vary rshpi dist=gauss std=rshpi_std // rshpi_std is a parameter
        vary xxx dist=lnorm std=12
        vary uuu dist=unif N=10 percent=yes
        ...
    }
    mismatch {  // mismatch: generate a random number per instance
        vary rshsp dist=gauss std=2
        vary xisn dist=gauss std=0.5
        vary xisp dist=gauss std=0.5
    }
    // some process parameters are correlated
    correlate param=[rshsp rshpi] cc=0.6
    // specify a global distribution truncation factor
    truncate tr=6.0     // +/- 6 sigma
}

//  a separate statistics block to specify correlated (i.e. matched)
//components

// where m1 and m2 are subckt instances.
statistics {
    correlate dev=[m1 m2] param=[xisn xisp] cc=0.8
}

//  a separate statistics block to specify correlation with wildcard, where
// `I*.M3' matches multiple subckt instances, for examples, I1.M3, I2.M3, I3.M3,
etc..
// Only the asterisk (*) is recognized as a valid wildcard symbol.

statistics {

    correlate dev=[ I*.M3 ] param=[misx mixy] cc=0.8

}
```

Note: You can specify the same parameter (for example, `rshsp`) for both process and mismatch variations.

In the process block, the process parameter `rshsp` is varied with a Gaussian distribution, where the standard deviation is 12 percent of the nominal value (`percent=yes`). When `percent` is set to `yes`, the value for the standard deviation (`std`) is a percentage of the nominal value. When `percent` is set to `no`, the specified standard deviation is an absolute number. This means that parameter `rshsp` should be varied with a normal distribution, where the standard deviation is 12 percent of the nominal value of `rshsp`. The nominal or mean

value for such a distribution is the current value of the parameter just before the Monte Carlo analysis starts. If the nominal value of the parameter `rshsp` was 200, the preceding example specifies a process distribution for this parameter with a Gaussian distribution with a mean value of 200 and a standard deviation of 24 (12 percent of 200). The parameter `rshpi` (sheet resistance) varies about its nominal value with a standard deviation of 0.4 K-ohms/square.

In the mismatch block, the parameter `rshsp` is then subject to *further* statistical variation on a per-subcircuit instance basis for on-chip variation. Here, it varies a little for each subcircuit instance, this time with a standard deviation of 2. For the first Monte Carlo run, if there are multiple instances of a subcircuit that references parameter `rshsp`, then (assuming `variations=all`) it might get a process random value of 210, and then the different instances might get random values of 209.4, 211.2, 210.6, and so on. The parameter `xisn` also varies on a per-instance basis, with a standard deviation of 0.5. In addition, the parameters `rshsp` and `rshpi` are correlated with a correlation coefficient (`cc`) of 0.6.

The `.mcdat` file, by default, displays the following statistics parameters in the Statistics section: `max`, `min`, `mean`, `variance`, `stddev`, `avgdev`, `avgdev`, and `failedtimes`. You can set the `mc_stat_list` option parameter to `all` to output all statistical parameters in the file.

The following is an example of the `.mcdat` file. The parameters in blue are added when you set the value of `mc_stat_list` to `all`.

```
Statistics:
max              1.68571e-08     27              4.42243e-09
min              1.39394e-08     27              4.03428e-09
mean             1.55027e-08     27              4.19754e-09
variance         3.36186e-19     0               7.31927e-21
stddev           5.79816e-10     0               8.55527e-11
avgdev           4.67287e-10     0               6.99047e-11
skewness         0.0611099       NaN             0.260798
kurtosis         -0.213995       NaN             -0.403713
Q1               1.50883e-08     27              4.13173e-09
median           1.55056e-08     27              4.19708e-09
Q3               1.58615e-08     27              4.25678e-09
CI_mean_2.5%     1.53877e-08     27              4.18057e-09
CI_mean_97.5%    1.56178e-08     27              4.21452e-09
CI_stddev_2.5%   5.09082e-10     0               7.51159e-11
CI_stddev_97.5%  6.73558e-10     0               9.93845e-11
failedtimes      0               0               0
```

Multiple Statistics Blocks

You can use multiple statistics blocks, which accumulate or overlay each other. Typically, process variations, mismatch variations, and correlations between process parameters are specified in a single statistics block. This statistics block can be included in a “process” `include` file, such as the ones shown in the example in “Process Modeling Using Inline Subcircuits” on page 110. A second statistics block can be specified in the main netlist where actual device instance correlations are specified as matched pairs.

The following statistics block can be used to specify the correlations between matched pairs of devices and probably is placed or included into the main netlist by the designer. These statistics are used in addition to those specified in the statistics block in the preceding section so that the statistics blocks “overlay” or “accumulate.”

```
// define correlations for "matched" devices q1 and q2
statistics {
        correlate dev=[q1 q2] param=[XISN...] cc=0.75
}
```

Note: You can use a single statistics block containing both sets of statements; however, it is often more convenient to keep the topology-specific information separate from the process-specific information.

Specifying Distributions

Parameter variations are specified using the following syntax:

```
vary PAR_NAME dist=type {std=<value> | N=<value>} {percent=yes|no}
```

Three types of parameter distributions are available: Gaussian, log normal, and uniform, corresponding to the *`type`* keywords `gauss`, `lnorm`, and `unif`, respectively. For both `gauss` and the `lnorm` distributions, you specify a standard deviation using the `std` keyword.

Note: You can also specify a global parameter as a value for `dist`. For example:

```
parameters DIST_snd=gauss
statistics {
    process {
        vary AGIDL_snd dist=DIST_snd std=1
    }
}
```

The following distributions (and associated parameters) are supported:

- Gaussian

This distribution is specified using `dist=gauss`. For the Gaussian distribution, the mean value is taken as the current value of the parameter being varied, giving a distribution denoted by Normal(mean,std). Using the example in Specifying Parameter Distributions Using Statistics Blocks parameter `rshpi` is varied with a distribution of Normal(5k,0.4k). The nominal value for the Gaussian distribution is the value of the parameter before the Monte Carlo analysis is run. The standard deviation can be specified using the `std` parameter. If you do not specify the `percent` parameter, the standard deviation you specify is taken as an absolute value. If you specify `percent=yes`, the standard deviation is calculated from the value of the `std` parameter multiplied by the nominal value and divided by 100; that is, the value of the `std` parameter specifies the standard deviation as that percentage of the nominal value.

Note: If `std=0` or `std=<param_name>=0` is found in the statistics block, Spectre generates an error. You can specify the `ignorezerovar=yes` parameter in the `options` statement to bypass this check.

- Log normal

  This distribution is specified using `dist=lnorm`. The log normal distribution is denoted by

  ```
  log(x) = Normal( log(mean), std )
  ```

  where x is the parameter being specified as having a log normal distribution.

  Note: log() is the natural logarithm function. For parameter `xxx` in the example in Specifying Parameter Distributions Using Statistics Blocks the process variation is according to

  ```
  log(xxx) = Normal( log(20000), 12)
  ```

  The nominal value for the log normal distribution is the value of the parameter before the Monte Carlo analysis is run. If you specify a normal distribution for a parameter `P1` whose value is 5000 and you specify a standard deviation of 100, the actual distribution is produced such that

  ```
  log(P1) = N( log(5000), 100)
  ```

- Uniform

  This distribution is specified using `dist=unif`. The uniform distribution for parameter `x` is generated according to

  ```
  x = unif(mean-N, mean+N)
  ```

  such that the mean value is the nominal value of the parameter `x`, and the parameter is varied about the mean with a range of ± N. The standard deviation is not specified for the uniform distribution, but its value can be calculated from the formula std=N/sqrt(3). The nominal value for the uniform distribution is the value of the parameter before the Monte Carlo analysis is run. The uniform interval is specified using the parameter `N`. For

example, specifying `dist=unif N=5` for a parameter whose value is 200 results is a uniform distribution in the range 200±`N`, that is, from 195 to 205. You can also specify `percent=yes`, in which case, the range is 200±`N`%, that is, from 190 to 210.

If the parameters max and min are specified, the nominal value is calculated as `(max+min)/2` and the uniform interval is calculated as `(max-min)/2`.

- Log uniform

  This distribution is specified using `dist=lunif`. The log uniform distribution is denoted by

  ```
  log(x) = (log(nominal_value) - N, log(nominal_value) + N)
  ```

  Here, `x` is the parameter being specified as having a log uniform distribution. The nominal value (`nominal_value`) is defined as `min*sqrt(max/min)` and the value of `N` is specified as `log(sqrt(max/min))`. The parameters `max` and `min` specify the distribution range.

Derived parameters that have their default values specified as expressions of other parameters cannot have distributions specified for them. Only parameters that have numeric values specified in their declaration can be subjected to statistical variation.

Parameters that are specified as correlated must have had an appropriate variation specified for them in the statistics block.

For example, if you have the parameters

```
XISN=XIS+XIB
```

you cannot specify distribution for `XISN` or a correlation of this parameter with another.

The `percent` flag indicates whether the standard deviation `std` or uniform range `N` are specified in absolute terms (`percent=no`) or as a percentage of the mean value (`percent=yes`). For parameter `uuu` in the example in Specifying Parameter Distributions Using Statistics Blocks the mean value is 200, and the variation is 200 ±10%*(200), that is, 200 ± 20. For parameter `rshsp`, the process variation is given by Normal(200, 12%*(200)), that is, Normal(200, 24). Cadence recommends that you do not use the `percent=yes` with the log normal distribution.

Changing Parameter Distributions at Runtime

At times, you might want to reproduce the design failures with less MonteCarlo iterations and check the design robustness quickly. You can use the `dist=default|unif|gauss` parameter to force all parameter distributions to a specified type. In addition, you can use the `stdscale` parameter scale the deviation by a specified value.

The following examples show how to use `stdscale` and `dist` options to force a parameter distribution to the specified type.

- `stdscale=k`

  Consider a scenario where you have the following in the statistics block:

  ```
  statistics {
  mismatch {
  vary mymismatch1 dist=gauss std=1
        }
  }
  ```

  If you specify `stdscale=k` in the MonteCarlo statement, `std=1` of all random variables will be multiplied by `k`. For example, if you specify the following in the Montecarlo statement:

  ```
  mc1 montecarlo firstrun=1 numruns=5000 dist=unif stdscale=2 {
  dc dc
  }
  ```

  the statistics block will change to:

  ```
  statistics {
  mismatch {
  vary mymismatch1 dist=gauss std=1*k
        }
  }
  ```

- `dist=unif|gauss`

  When you specify `dist=unif` or `dist=gauss`, distribution of all random variables is converted to `unif` or `gauss`.

  Consider a scenario where you have the following in the statistics block:

  ```
  statistics {
  mymismatch1 = p
  mismatch {
  vary mymismatch1 dist=gauss std=a
  }
  }
  ```

  If you specify the following MonteCarlo statement:

  ```
  mc1 montecarlo numruns=100 variations=mismatch seed=12345 stdscale=k dist=unif
  ```

  the statistics block will change to:

  ```
  statistics {
  mymismatch1 = p
  ```

```
mismatch {
vary mymismatch1 dist=unif N=a*sqrt(3)*k
}
}
```

Note: When `stdscale=k` and `dist=unif`, is specified and if one parameter follows `gauss`, `std=a`, `mean=b`, Spectre automatically converts it into uniform distribution with `mean=b`, `unif(b-sqrt(3)*a*k, b+sqrt(3)*a*k)`.

Note: By transforming the Gaussian distribution to Uniform, you can identify the design failure at high sigma-corner with small number of samples. However, you cannot predict the yield of the design with this approach.

Truncation Factor

The default truncation factor for Gaussian distributions (and for the Gaussian distribution underlying the log normal distribution) is 4.0 sigma. Randomly generated values that are outside the range of mean $\pm$ 4.0 sigma are automatically rejected and regenerated until they fall inside the range. If the truncation factor is less than 0, Spectre does not generate truncated distributions and generates a warning. If the truncation factor is specified as 0, then Spectre generates an error.

You can change the truncation factor using the `truncate` statement. The following is the syntax:

```
truncate tr=value
```

The value of the truncation factor can be a constant or an expression. In addition, you can specify the truncation factor in the process and mismatch blocks. However, if the truncation factor is not specified in the process or mismatch block, then the truncation factor in the statistics block is considered.

Note: Parameter correlations can be affected by using small truncation factors.

Correlation Statements

There are two types of correlation statements that you can use:

- Process parameter correlation statements

  The following is the syntax of the process parameter correlation statement:

  ```
  correlate param=[list of parameters] cc=value
  ```

  This allows you to specify a correlation coefficient between multiple process parameters. You can specify multiple process parameter correlation statements in a statistics block to

build a matrix of process parameter correlations. During a Monte Carlo analysis, process parameter values are randomly generated according to the specified distributions and correlations.

- Instance or mismatch correlation statements (matched devices)

  The following is the syntax of the instance or mismatch correlation statement:

  ```
  correlate dev=[list of subckt instances] {param=[list of parameters]}
  cc=value
  correlate dev=[<wildcard expr>] {param=[list of parameters]} cc=<value>
  ```

  where the device or subcircuit instances to be matched are listed in *list of subckt instances*, or regular expressions with asterisk (*) and *list of parameters* specifies exactly which parameters with mismatch variations are to be correlated. Use the instance mismatch correlation statement to specify correlations for particular subcircuit instances. If a subcircuit contains a device, you can effectively use the instance correlation statements to specify that certain devices are correlated (matched) and give the correlation coefficient. You can optionally specify exactly which parameters are to be correlated by giving a list of parameters (each of which must have had distributions specified for it in a mismatch block) or by specifying no parameter list, in which case all parameters with mismatch statistics specified are correlated with the given correlation coefficient. The correlation coefficients are specified in the <*value*> field and must be between ± 1.0.

Note: Correlation coefficients can be constants or expressions, as can `std` and `N` when specifying distributions.

Characterization and Modeling

The following statistics blocks can be used with the example in "Process Modeling Using Inline Subcircuits" on page 110 if they are included in the main netlist, anywhere below the main `parameters` statement. These statistics blocks are meant to be used in conjunction with the modeling and characterization equations in the inline subcircuit example, for a Monte Carlo analysis only.

```
statistics {
    process {
       vary RSHSP dist=gauss std=5
       vary RSHPI dist=lnorm std=0.15
       vary SPDW dist=gauss std=0.25
       vary SNDW dist=gauss std=0.25
    }
    correlate param=[RSHSP RSHPI] cc=0.6
    mismatch {
       vary XISN dist=gauss std=1
       vary XBFN dist=gauss std=1
       vary XRSP dist=gauss std=1
    }
}
```

```
statistics {
    correlate dev=[R1 R2] cc=0.75
    correlate dev=[TNSA1 TNSA2] cc=0.75
}
```

Creating same variation for same block in different simulations

At times, it may be necessary to use the same variation of a design in different simulations. This can be achieved by using identical subcircuit and instance definitions for the design in both simulations, and by ordering the netlist with the `options` statement, as shown below.

```
opt1 options sortinstance=yes
```

Distributed Monte Carlo Analysis

If a Monte Carlo analysis is defined in the netlist and the `+mp <numprocesses>` command-line option is used, Spectre FX automatically detects the farm environment (LSF, SGE, RTDA, or Network Computer) and distributes the montecarlo analysis to the specified number of child processes. If a farm environment is not detected, Spectre FX uses the `fork` option to distribute a Monte Carlo analysis by creating multiple jobs on a single system.

Spectre FX supports distribution of Monte Carlo and Sweep analyses; however, any subset analysis defined within the Monte Carlo or Sweep analysis is not distributed.

Note: You cannot use relative path (such as `scalarfile=../monteCarlo/mcdata`) to specify the location of file in distributed montecarlo. This is because distributed results are saved in the root directory. If you use a relative path to specify the location, all child processes will refer to the same directory.

Reliability Analysis

Spectre reliability analysis for Hot-Carrier Injection (HCI), Negative Bias Temperature Instability (NBTI), and Positive Bias Temperature Instability (PBTI) modules is a two-phase simulation flow. The first phase, fresh and stress simulation, calculates the device age or degradation. The second phase, post-stress or aging simulation, simulates the degradation effect on the circuit performance based on the device degradation information obtained during the first phase of stress simulation.

The syntax is as follows:

```
Name reliability [global_options]{
   reliability control statements
   stress statements
   aging/post-stress statements
}
```

Reliability Parameters Supported by Spectre FX

The following table describesthe reliability parameters supported with Spectre FX.

Parameter	Description
`age time`	The time in future when the transistor degradation and degraded SPICE model parameters are to be calculated.
`report_model_param value`	Specifies whether to print the stress and aged parameters in the.bm# file. Possible values are `no` and `yes`.
`accuracy level`	Specifies the methods used in the reliability simulation when performing integration and substrate current calculation. Possible values are `1` and `2`.
`minage value`	Specifies the smallest age value for which degraded SPICE model parameters are calculated.
`igatemethod type`	Specifies the method used for obtaining the gate currents of MOSFETs. Possible values are `calc` and `spice`.

Parameter	Description
`isubmethod type`	Specifies the method used for obtaining the substrate currents of MOSFETs. Possible values are `calc` and `spice`.
`opmethod type`	Specifies whether the Igate or Isub value is obtained from the SPICE models, such as BSIM3 and BSIM4 or from the internal Igate or Isub equation. Possible values are `calc` and `spice`.
`enable_negative_age value`	Enables negative age value. Possible values are `no` and `yes`.
`idmethod type`	Specifies how the simulator obtains the drain current (Id) of MOSFETs to perform reliability calculations. Possible values are `ids`, `idrain` and `idstatic`.
`enable_ade_process value`	Enables reliability analysis in ADE Explorer or ADE Assembler. Possible values are `no` and `yes`.
`tmi_she_mindtemp value`	Specifies the minimum delta temperature for self-heating flow.
`rel_mod type`	Specifies the analysis type used for obtaining the device reliability values. Possible values are `aging`, `she`, `all`, `aging_she` and `aging_thermal`.
`relxtran start`	Specifies the start time of reliability analysis during transient simulation.
`urilib file`	Specifies the name of the URI library file.
`urilib uri_mode`	Specifies the method to be used for performing aging simulation. Possible values are `agemos`, `scaleparam`, `appendage`, `new_appendage`, `appendage2` and `appendage1`.

Parameter	Description
`urilib debug`	Specifies the debug mode for URI library. The value can be either `0` or a positive value. When specified, a flag is added to the URI library indicating whether to print the debug information. If `debugMode` is not set to `0`, the debug messages are printed. Default value is `0`.
`urilib scale_mode`	Specifies parameter scale mode for URI library. The value can be `original` or `effective`. If value is `original`, send original parameter value to URI for param scale, else send effective value. Possible values are `original` and `effective`.
`tmi_aging_mode`	Specifies the mode of TMI aging flow. Possible values are `aging`, `she`, and `all`.

Reliability Models for Spectre FX

Spectre FX supports the following reliability models:

- Agebsim4
- Agebsim3v3
- Agebsimsoi
- Agepsp102
- Agehisim2
- Agehisim_hv
- Agebsimcmg
- Ageumos3
- Ageumos4
- Ageumos5
- Ageumos6
- Agepsp103

- Agepsp1020
- Agepsp1021
- Agepsp102e
- Agetmibsimcmg
- Agebsimimg
- Ageutsoi
- Ageutsoi2
- Agelutsoi
- Agetmibsim6
- Agetmibsimbulk
- Agebsim6
- Agesimbulk
- Agehisimhv_va
- Agemos195
- Agesisim2_va

Reliability Feature Support Matrix for Spectre FX

This matrix applies to SPECTRE 23.1 and subsequent ISR versions.

Flow Name	Two Netlist Flow				One Netlist Flow
	Age only	SHE only	RT SHE	AGE + SHE	
Appendage	Yes	No	No	No	No
TMI	Yes	Yes	Yes	Yes	No
EM-IR	No	Yes	Yes	No	No

Note that the `maskdev` control statement is not supported in Spectre FX.

5

Spectre FX Simulator Options

This topic describes the options that are specific to the Spectre FX simulator and cannot be used by other simulators on the Spectre platform.

Related Topics

Specifying Command-Line Options for Spectre FX

Loading Spectre FX Options from a Configuration File

Specifying Spectre FX Options with the options or .option Argument

Setting the Spectre FX Options Locally

Setting Up Different Speeds for Specific Periods of Transient Simulations

Viewing the Options in Log Files

Spectre FX Options for the options or .option Argument

Specifying Command-Line Options for Spectre FX

You can set the simulation preset mode or the high-level options at the command line. The command-line options help you to further simplify the use model.

You can specify the following command-line options for Spectre FX:

Option	Description
`[preset=mode]`	Specifies the preset mode. Possible values are `baseax`, `basemx`, `baselx`, `basevx`, `ax`, `mx`, `lx`, and `vx`. The default mode is `lx`.
`+[option]`	Specifies the additional options that can be used. Possible options are `speed` and `optimize`.

Note: The options specified at the command line have the highest priority and override the same setting in the configuration file or the netlist.

Examples

The following code sets the preset mode to `baselx` and the parasitic optimization to `mx`:

```
%> spectrefx +baselx +optimize=mx -f fsdb +mt=8 top.sp
```

The following code sets the `speed` option for simulation tolerance to `mx`. By default, the preset mode is `lx`.

```
%> spectrefx +speed=mx +mt=8 top.sp
```

Related Topics

Specifying Spectre FX Options with the options or .option Argument

Loading Spectre FX Options from a Configuration File

Loading Spectre FX Options from a Configuration File

Spectre FX can read the simulation configuration files containing a set of options.

You can load configuration files in one of the following ways:

- **spectre.cfg configuration file:** A predefined file name, `spectre.cfg`, can be used to create a configuration file. This file can be loaded to set the default options for the simulator. This file is located and automatically read from the following locations:
  - The installation directory `<install_dir>/spectre/etc/configs`. This file is always read.
  - One of the following locations, in the given order of precedence:
    - Working directory
    - Home directory

    Therefore, if a configuration file exists in both the working directory and the home directory, Spectre reads the one from the working directory.
- **Any other configuration file:** You can also specify the options in a configuration file that can be added using the `+config` command-line option. If you specify multiple files using multiple `+config` command-line options, all files are read by Spectre FX. In this way, the original netlist need not be changed for the Spectre FX simulation.

```
%> spectrefx +config /path/user1.cfg +config /path/user2.cfg -f fsdb +mt=8
top.sp …
```

If `+echooptions` is specified at the command line, Spectre FX displays how the configuration files are read. It also displays the options from the configuration files.

Important

Options in the configuration file added using the `+config` on command-line option, have higher priority than the options specified in the netlist and `spectre.cfg` file.

Related Topics

Specifying Command-Line Options for Spectre FX

Specifying Spectre FX Options with the options or .option Argument

Specifying Spectre FX Options with the options or .option Argument

Spectre FX supports both Spectre and SPICE syntax to set the options.

To specify a Spectre FX option, use any one of the following syntax:

Spectre syntax

SimOpt1 options <*option_name1*>=<*option_value*> <*option_name2*>=<*option_value*> …

SPICE syntax

.option <*option_name1*>=<*option_value*> <*option_name2*>=<*option_value*> …

Note: With the exception of the backannotation flow options, you can set the options multiple times. However, only the option that is specified last is considered. Spectre FX shows a warning message for the options that are specified more than once.

Related Topics

Setting the Spectre FX Options Locally

Spectre FX Options for the options or .option Argument

Setting the Spectre FX Options Locally

By default, Spectre FX options are applied at the global level. However, you can set the following options locally for a subcircuit by using the `subckt=` option or for specific instances by using the `inst=` option. Wildcards are supported for `subckt` and `inst`.

- sfx_preset
- sfx_speed
- sfx_speed_window

Examples

Sets the option locally on subcircuit primitives:

```
.option sfx_speed=mx subckt=pll
.option sfx_preset=basemx subckt=[osc bandgap]
```

Sets the option locally on instances:

```
.option sfx_speed=lx inst=x1.x1
.option sfx_speed=ax inst=[x1.x2 x1.x3]
```

In post-layout simulations with flatten DSPF file, there is no design hierarchy. `sfx_preset` can apply to specific elements in a DSPF file, as shown below.

```
.option sfx_preset=baseax subckt=[pll] element=[*osc*]
.option sfx_preset=baseax subckt=[pll] element=[x1/x2/xosc*]
```

In these examples, `pll` is the subckt name defined in a DSPF file. `*osc*` or `x1/x2/xosc*` is the instance name in DSPF file. `element=[*osc*]` or `element=[x1/x2/xosc*]` applies the local option to all MOSFETs in the specified oscillator block.

Related Topics

Specifying Spectre FX Options with the options or .option Argument

Setting Up Different Speeds for Specific Periods of Transient Simulations

For a transient simulation, a design may have different accuracy requirements for the whole period. For example, a Flash design in power-up stage might have higher tolerance for the accuracy margin than read/program/erase operation modes.

To better tune accuracy and performance, Spectre FX provides the `sfx_speed_window` option for tuning based on different transient times.

Example

```
.option sfx_speed_window=[0 lx 50u ax]
```

Sets different speeds for specific periods of the transient simulation.

Related Topics

sfx_speed_window

Viewing the Options in Log Files

The options specified in Spectre FX are displayed in the log file.

A Spectre FX log file displays the following information:

- The command-line options are displayed at the beginning of the log file. For example:

```
/vols/mmsim/19.10/amazon/install/11-11-2020/tools.lnx86/bin/spectrefx  \
-64 +baselx -f fsdb top.sp -outdir sfx_results/11-11-2020/blx_8T +mt=8
```

- Warning message for an unrecognized option or wrong value of the supported options. There are also warning messages for the duplicate options set.
- If `+echooptions` is specified at the command line, Spectre FX displays the options categorized by files.
- The final valid options are listed under the `Global user options:` and `Scoped user options:` sections.

Related Topics

Reviewing the Log File

Spectre FX Options for the options or .option Argument

The Spectre FX options can be categorized as follows:

- High-Level Accuracy/Performance Tuning Options
- Solver Related Options
- RC Tuning Options
- Device Model Options
- Parsing Options
- Probing and Measurement Options
- Backannotation Flow Options
- Miscellaneous Options

Note: Options names that begin with `sfx_` are proprietary to Spectre FX. The other options can be shared with other Spectre simulators.

High-Level Accuracy/Performance Tuning Options

The following table lists the accuracy/performance tuning options:

Option	Description
`sfx_optimize`	Defines the parasitic optimization, including RC Reduction. Possible values are `ax`, `mx`, `lx`, and `vx`. The option can be set locally
`sfx_preset`	High-level preset for accuracy/performance tradeoff. Possible values are `baseax`, `basemx`, `baselx`, `basevx`, `ax`, `mx`, `lx`, and `vx`. This option can be set locally.
`sfx_speed`	Defines the simulation tolerance for the voltage and current calculation. The option can be set locally.

Solver Related Options

The following table lists the options related to resistors and capacitors:

Option	Description
sfx_integ_method	Defines the integration method for transient simulation. Possible values are `gear`, `gear2`, `traponly` and `euler`.
sfx_speed_signal	Defines the simulation tolerance only for the signal nodes that are detected automatically by Spectre FX.
sfx_speed_power	Defines the simulation tolerance only for the power nets that are detected automatically by Spectre FX.
sfx_speed_window	Sets the option `sfx_speed` based on the time window during the transient simulation.
sfx_maxstep_window	Sets the transient simulation control parameter `maxstep` based on the time window.
sfx_time_step_errtol_factor	Specifies the factor to tune the tolerance of time step.
sfx_unit_time	Specifies the transient simulation time unit.

RC Tuning Options

The following table lists the options related to resistors and capacitors:

Option	Description
sfx_ccap_cut	Defines the threshold value for the capacitors that need not be cut during partitioning.
sfx_ccap_esv	Defines the voltage threshold of the event related to the capacitors.
sfx_optimize_power	Defines the parasitic optimization only for the power nets that are detected automatically by Spectre FX.
sfx_optimize_signal	Defines the parasitic optimization only for the signal nodes that are detected automatically by Spectre FX.
sfx_rcr	Defines RC Reduction, which is controlled by `sfx_optimize` if no definition is given.
sfx_rmin	Resistors with value less than the specified value are shorted.

sfx_rmax	Resistors with value more than the specified value are open.

Device Model Options

The following table lists the options related to device models:

Option	Description
cmi_mosfet	Lists the device names that are treated as mosfet.
sfx_support_nqs	Supports `trnqsmod` of bsim4 and bsim3v3, if it is defined as `1`.
sfx_fast_mosfet_cap	Specifies whether to enable the fast calculation for MOSFET capacitances.
soft_bin	If set to `singlemodels`, it is used only on non-binned models. Possible values are `singlemodels`, `no`, and `allmodels`.

Parsing Options

Spectre FX supports both SPICE syntax and Spectre syntax for most of the options and the related parsing options. The following table lists only a few selected options.

Option	Description
duplicate_measure	Specifies whether measures are allowed.
duplicate_module	Specifies whether duplicate module definitions are allowed.
duplicate_subckt	Specifies whether duplicate subcircuit definitions are allowed.
duplicateinstance	Specifies whether duplicate instance definitions are allowed.
duplicatemodel	Specifies whether duplicate model definitions are allowed.
duplicateports	Specifies whether duplicate ports are allowed in the subcircuit definition.
redefinedparams	Specifies whether parameters can be redefined in the netlist.
skip	Defines the simulation tolerance for the voltage and current calculation. This option can be set locally.

Note: For detailed information, refer to *Spectre Classic Simulator, Spectre APS, Spectre X, and Spectre XPS User Guide*.

Probing and Measurement Options

The following table lists the options to control output statements and output format.

Option	Description
dotprobefmt	Prints the `.probe` signal with the original name or the hierarchical name. Possible values are `flat` and `hier`.
mt_format	Specifies the format of the measurement file.
mt_separate_sweep_files	Specifies whether to separate or combine the measurement files for sweep analyses.
output_case	Outputs all signal names in lower case or upper case. Possible values are `original`, `lower`, and `upper`. The default value is `original`.
probe_global_term	Probes the implicit global terminal of the subcircuit which does not exist in the netlist but is drilled because the global node is referred to in the subcircuit. Possible values are `no` and `yes`. The default value is `no`.
probe_level	Specifies how to count hierarchical depths for probe statement.
sfx_compress_waveform	High-level option for waveform compression.
sfx_compress_waveform_iabstol sfx_compress_waveform_ireltol sfx_compress_waveform_vabstol sfx_compress_waveform_vreltol	Detailed options for waveform compression. Separate options are provided to further tune the absolute and relative error tolerance for voltage and current signals.
sfx_lprobe_format	Specifies the location where you want to save the logic waveforms.

sfx_separate_current_waveform	Specifies whether to save a separate waveform file for current signals. By default, Spectre FX saves both voltage and current signals to the same waveform file.
sfx_waveform_flush_percentage	Defines the percentage of transient simulation to dump the waveforms to the file.
sfx_waveform_flush_minute	Defines the specified period of transient simulation to dump the waveforms to file.

Backannotation Flow Options

The control options for the parasitic backannotation flow are discussed in the section Control Options for the Parasitic Backannotation Flow.

Miscellaneous Options

Option	Description
print_section	Prints the information about the library files read during the simulation.
sfx_cshunt	Specifies the value of grounded capacitors to be inserted for each nodes.
sfx_cut_floating_gate	Adjusts the partition algorithm to perform the cut for the floating gate of the MOSFET.
sfx_dump_floating_gate_node	Specifies whether to dump all the detected floating gate nodes.
sfx_floating_gate_gshunt	Specifies the value of `gshunt` placed on floating gate nodes.
sfx_floating_node_gshunt	Specifies the value of `gshunt` placed on floating nodes without conducting path to both VDD and Ground.
sfx_oscillator_node	Specifies one node of an oscillator loop. The simulator automatically starts the oscillation when the oscillator is enabled.
sfx_progress_percentage	Specifies the frequency to update the transient status based on the percentage value.

cmi_mosfet

Syntax

```
cmi_mosfet = [val1 val2]
```

Description

Specifies a list of device names that are treated as MOSFETs. This option is used for the customized CMI devices for the MOSFET.

dotprobefmt

Syntax

```
dotprobefmt = flat | hier
```

Description

Print the `.probe` signal with the original name or the hierarchical name. Possible values are `flat` and `hier`.

duplicate_subckt

Syntax

```
duplicate_subckt = error | warning | ignore
```

Description

Specifies whether duplicate subcircuit definitions are allowed. When this option is set to `warning` or `ignore`, the simulator allows duplicate subcircuit definitions. If multiple subcircuit definitions are specified, the last one takes precedence. Possible values are `error`, `ignore`, and `warning`. The default value is `error`.

duplicateports

Syntax

```
duplicateports = error | warning | ignore
```

Description

Specifies whether duplicate ports are allowed in the subcircuit definition. When this option is set to `warning` or `ignore`, the duplicate ports are shorted. Depending upon the value set, the simulator displays warning messages for duplicate ports, or does not display any message. When set to `error`, the simulator does not allow duplicate ports in the subcircuit definition and displays an error message. Possible values are `error`, `ignore`, and `warning`. The default value is `error`.

duplicatemodel

Syntax

```
duplicatemodel = error | warning | ignore
```

Description

Specifies whether duplicate model definitions are allowed. When set to `warning` or `ignore`, the simulator allows duplicate model definitions. However, it honors only the last model definition. Depending upon the value set, the simulator displays warning messages for duplicate model definitions, or does not display any message.

When set to `error` for a model in Spectre format, the simulator does not allow duplicate model definitions and displays an error message. For a model in SPICE format, the model definition is overridden and the simulator does not display an error message even if `duplicatemodel` is set to `error`.

Possible values are `error`, `ignore`, and `warning`. The default value is `error` for models in Spectre format, and `warning` for models in SPICE format.

duplicate_measure

Syntax

```
duplicate_measure = error | first_win | last_win
```

Description

Specifies whether duplicate measures are allowed. When set to `first_win`, the simulator allows duplicate measures. However, it honors only the first measure. When set to `last_win`, the simulator honors the last measure. When set to `error`, the simulator does not allow duplicate measures and displays an error message. Possible values are `first_win`, `last_win`, and `error`. The default value is `error`.

duplicateinstance

Syntax

```
duplicateinstance = error | warning | ignore
```

Description

Specifies whether duplicate instance definitions are allowed. When this option is set to `warning` or `ignore`, the simulator allows duplicate instance definitions. If multiple instance definitions are specified, the last one takes precedence. Possible values are `error`, `ignore`, and `warning`. The default value is `error`.

duplicate_module

Syntax

```
duplicate_module = error | warning | ignore
```

Description

Specifies whether duplicate module definitions are allowed. When set to `warning` or `ignore`, the simulator allows duplicate module definitions. However, it honors only the last module definition.

Depending upon the value set, the simulator displays warning messages for duplicate module definitions, or does not display any message. When set to `error`, the simulator does not allow duplicate module definitions and displays an error message. Possible values are `error`, `ignore`, and `warning`. The default value is `warning`.

mt_format

Syntax

```
mt_format = horizontal | vertical
```

Description

You can use the `mt_format` option to control the format of the `mt0` file. Possible values are `horizontal` and `vertical`.

- `horizontal`: Keeps the original format of the `mt0` file. This is the default value.
- `vertical`: Each measure value is specified in a separate line and the format is `meas_name=`*`val`*.

Examples

A sample `mt0` file with the option `mt_format=horizontal`:

```
index temp x_load tr2f tf2r cload_i temper alter#
1 -40 1e-13 5.73137e-09 4.69658e-09 -3.9815e-11 -40 2
2 -40 2e-13 9.47969e-09 7.47218e-09 3.07635e-08 -40 2
……
```

A sample `mt0` file with the option `mt_format=vertical`:

```
index = 1
temp = -40
x_load = 1e-13
tr2f = 5.73137e-09
……


index =2
temp = -40
x_load = 2e-13
tr2f = 9.47969e-09
……
```

mt_separate_sweep_files

Syntax

```
mt_separate_sweep_files = value
```

Description

Specifies whether to separate or combine the measurement files for sweep analyses.

output_case

Syntax

```
outputcase = original | lower | upper
```

Description

Output all signal names in lower case or upper case. Possible values are `original`, `lower`, and `upper`. The default value is `original`.

print_section

Syntax

```
print_section = yes | no
```

Description

Enables you to print the information about the library files read during the simulation. Possible values of `no` and `yes`:

- `no`: Only the library file information is printed in the log file.
- `yes`: Detailed sections of the library files read during the simulation are printed in the log file.

Example

If you specify `print_section=yes`, the log file displays the following information for the same example:

```
Reading file:  /projects/pl00/simulation/amazon_run/top.sp
Reading file:  /projects/p100/simulation/amazon_run/netlist.cdl
Loading section: SS from file: /projects/pl00/simulation/amazon_run//PDK/Spectre/
v1.1/16nm model.scs
Loading section: TT BIP DIO from file: /projects/pl00/simulation/amazon_run//PDK/
Spectre/vl.1/16nm_model.scs
Loading section:TT RES DISRES from file: /projects/pl00/simulation/amazon_run//
PDK/Spectre/v1.1/16nm_model.scs
```

probe_global_term

Syntax

```
probe_global_term = 0 | 1
```

Description

If `probe_global_term = 1`, Spectre FX probes the implicit global terminal of the subcircuit that does not exist in netlist but is drilled because the global node is referred to in the subcircuit. Possible values are `0` and `1`. The default value is `0`.

probe_level

Syntax

```
probe_level = wildcard | top
```

Description

Specifies how to decide the hierarchical depth. Possible values are `top` and `wildcard`.

- `top`: Count the level starting from the top level.
- `wildcard`: The level/depth specified in the `.probe` statement is applied to wildcard(s) only. The hierarchical delimiters explicitly specified in `.probe` extend the level/depth to be probed (default).

Example

```
.option probe_level=wildcard
.probe v(x1.XIdecod4.*) level=1
.probe v(x1.XIdecod3.*) preserve=all level=2
.probe v(x1.XIdecod2.*) except=[*net*] level=3
```

In the Spectre FX log file, the matching of wildcard is given as:

```
Wildcard match summary:
     probe v(x1.XIdecod3.*) depth= 2: 1098
     probe v(x1.XIdecod2.*) depth= 3: 206
     probe v(x1.XIdecod4.*) depth= 1: 16
```

Here, the option `probe_level` is defined as `wildcard`. As a result, the depth specified with the `level` parameter is applied to only the wildcard. The parameter `preserve=all` in the second `.probe` statement enables Spectre FX to probe all the nodes in different hierarchical levels and parameter `except` in the third `.probe` statement excludes all the nodes with the name `*net*`.

redefinedparams

Syntax

```
redefinedparams= error | warning | ignore
```

Description

Specifies whether parameters can be redefined in the netlist. If a parameter is redefined multiple times, the last parameter definition takes precedence. Possible values are `error`, `warning`, and `ignore`. The default value is `warning` in SPICE compatibility mode (`+spice`) and `error` in Spectre mode.

sfx_accelerate_model_eval

Syntax

```
sfx_accelerate_model_eval = 0 | 1
```

Description

Specifies whether to disable the model evaluation accelerator for MOSFETs. This option is mostly used for debugging purposes.

sfx_ba_bus_delimiter

Syntax

```
sfx_ba_bus_delimiter="char"
```

Description

Defines the bus delimiter in the DSPF file.

Example

```
.option sfx_ba_bus_delimiter = "[ ]"
```

sfx_ba_ccap_min

Syntax

```
sfx_ba_ccap_min
```

Description

Splits and grounds the coupling capacitors whose value is less than the specified value during the backannotation stage. The default value is 0 F, which means that no coupling capacitors are split and grounded.

sfx_ba_ccap_net

Syntax

`sfx_ba_ccap_net=[`*net1 … netN*`]`

Description

Backannotates the coupling and ground capacitances of the specified nets. All the resistors of the NET are shorted.

sfx_ba_ccap_net_file

Syntax

`sfx_ba_ccap_net_file=`*`filename`*

Description

Specifies a text file, which contains the list of nets whose total coupling and ground capacitance needs to be backannotated.

sfx_ba_dspf

Syntax

```
sfx_ba_dspf=filename subckt=[]
```

Description

Specifies the DSPF file with both `RC NET` section and `Instance` section to be read.

If no `subckt=[]` is specified in this statement, Spectre FX matches the subcircuit in the parasitic file with the same name in the pre-layout. If no subcircuit is matched, the behavior of Spectre FX is decided by the `sfx_ba_unrecognized_subckt_action` option. You can explicitly specify the subcircuit or instance name in the pre-layout using the `subckt=[]` parameters.

sfx_ba_dspf_instance

Syntax

```
sfx_ba_dspf_instance=file subckt=[]
```

Description

This option is similar to the `sfx_ba_dspf_net` option; however, it reads the part of the Instance section.

For the same subcircuit or instance, you can specify different files for the RC Net section and Instance section.

If a DSPF file contains both the RC NET and Instance sections to be backannotated, you need to specify the same file using the `sfx_ba_dspf_net` and `sfx_ba_dspf_instance` options.

sfx_ba_dspf_net

Syntax

`sfx_ba_dspf_net=`*`file`* `subckt=[]`

Description

Specifies the DSPF file or the SPF file to be read. For each option, only one file can be specified. You need to define multiple options to read more than one DSPF file.

For the file invoked by `sfx_ba_dspf_net=`, Spectre FX reads only the RC Net part and ignores the Instance section in this file.

If no `subckt=[]` is specified in this statement, Spectre FX matches the subcircuit in the parasitic file with the same name in the pre-layout. If no subcircuit is matched, Spectre FX behavior is decided by the sfx_ba_unrecognized_subckt_action option. You can explicitly specify the subcircuit or instance name in the pre-layout using the `subckt=[]` parameters.

sfx_ba_finger_delimiter_string

Syntax

```
sfx_ba_finger_delimiter_string="string"
```

Description

Defines the finger delimiter string for instances in the DSPF file.

sfx_ba_gcap_net

Syntax

`sfx_ba_gcap_net=[`*`net1 … netN`*`]`

Description

Backannotates the total ground capacitance of the specified nets.

Use the option `sfx_ba_ground_dangling_ccap=1` together with `sfx_ba_gcap_net` to keep coupling capacitors that connect to `gcap_net`.

sfx_ba_gcap_net_cth

Syntax

`sfx_ba_gcap_net_cth=`*`val`*

Description

Backannotates the total ground capacitance of the net whose total capacitance is smaller than the specified value.

Use the option `sfx_ba_ground_dangling_ccap=1` together with `sfx_ba_gcap_net_cth` to keep the coupling capacitors that connect to `gcap_net`.

sfx_ba_gcap_net_file

Syntax

`sfx_ba_gcap_net_file=`*`filename`*

Description

Specifies a text file, which contains the list of nets whose ground capacitances need to be backannotated.

sfx_ba_ground_dangling_ccap

Syntax

```
sfx_ba_ground_dangling_ccap=0 | 1
```

Description

If set to `1`, coupling capacitors with an invalid node connection are grounded. If set to `0`, the invalid node connections are dropped. If both nodes are invalid, the capacitor is dropped in all the scenarios. The default value is `0`.

sfx_ba_hier_delimiter

Syntax

```
sfx_ba_hier_delimiter = Char
```

Description

Defines the hierarchical delimiter in the DSPF file. If `*|divider` is specified in the DSPF file, it has a higher priority. This option works only if `*|divider` is not specified in the DSPF file.

sfx_ba_instance_scale

Syntax

`sfx_ba_instance_scale=`*`val`*

Description

This option is used to scale the geometry of devices in the Instance section of the device, which can be different from the scaling in pre-layout.

sfx_ba_probe_node

Syntax

`sfx_ba_probe_node=[`*`val`* `net_name:`*`val`*`]`

Description

Specifies how to probe the nodes in an RC NET. The net name with a wildcard can be specified to localize this option. If the NET name is not specified, it is considered global.

Possible values are `rep`, `pin`, `all`, `gate`, and `primary_gate`.

- `rep`: Do not probe backannotated nodes. This is the default.
- `pin`: Probe the instance pins of the SPF net.
- `all`: Probe all instance pins and internal nodes of the SPF net.
- `gate`: Probe the instance pins of all device gates
- `primary_gate`: Probe the instance pins of the primary device gate (skip finger devices).

sfx_ba_probe_node_name

Syntax

```
sfx_ba_probe_node_name=full | brief
```

Description

Specifies how to define the signal name if you set the sfx_ba_probe_node option to a value other than `rep` because the backannotated nodes need to be probed. The default value is `brief`.

- `brief`: The probed backannotated nodes use the name in the DSPF file as the signal name.
- `full`: The pre-layout node name and the character `#` is added as the prefix for the signal name.

For example, if a pre-layout node `x1.mid` has a pin `xp1/mp:d`, and you want to probe this pin as `v(x1.mid#xp1/mp:d)`

sfx_ba_rc_net

Syntax

`sfx_ba_rc_net=[`*net1 … netN*`]`

Description

Backannotates the RC tree of the specified nets as defined in the RC NET section.

sfx_ba_rc_net_file

Syntax

`sfx_ba_rc_net_file=`*`filename`*

Description

Specifies a text file that contains the list of nets whose RC tree needs to be backannotated.

sfx_ba_remove_prefix

Syntax

```
sfx_ba_remove_prefix =[prefix1 prefix2 … prefixN]
```

Description

This is a string vector option. You can use one or more of these options to specify the characters of an instance to be removed so that the post-layout element can match the one in the pre-layout. There is no default value.

sfx_ba_rep_pin

Syntax

```
sfx_ba_rep_pin=first_pin | last_pin | first_gate | last_gate | first_gate/first_pin
     | first_gate/last_pin | last_gate/first_pin | last_gate/last_pin
```

Description

Specifies how to choose the representative node for an RC NET. Possible values are:

- `first_pin`: The first instance pin listed in `*|I`.
- `last_pin`: The last instance pin.
- `first_gate`: The first gate of MOSFETs.
- `last_gate`: The last gate of MOSFETs.
- `first_gate/first_pin`: If no gate is specified, it equals to `first_pin`.
- `first_gate/last_pin`: if no gate is specified, it equals to `last_pin`.
- `last_gate/first_pin`: If no gate is specified, it equals to `first_pin`.
- `last_gate/last_pin`: If no gate is specified, it equals to `last_pin`.

sfx_ba_rmin

Syntax

`sfx_ba_rmin=`*`val`*

Description

Shorts the resistor during the backannotation stage. The default value is 0.1 ohm.

sfx_ba_skip_net

Syntax

`sfx_ba_skip_net=[`*`net1 … netN`*`]`

Description

Specifies the nets that are to be skipped for backannotation, that is, none of the parasitic components of the nets are backannotated. If there is a coupling capacitor connected to net A and net B, and net A is skipped, this coupling capacitor is grounded and connected to net B only. Wildcard is supported in net names.

Use the option `sfx_ba_ground_dangling_ccap=1` together with `sfx_ba_skip_net` to keep the coupling capacitors that connect to `skip_net`.

sfx_ba_skip_net_file

Syntax

`sfx_ba_skip_net_file=`*`filename`*

Description

Specifies a text file, which contains the list of nets to be skipped for backannotation.

sfx_ba_unrecognized_subckt_action

Syntax

```
sfx_ba_unrecognized_subckt_action=error | ignore |top
```

Description

Specifies how to handle a DPSF file whose subcircuit name does not match any of the subcircuit names in the pre-layout. Possible values are:

- `error`: Generates an error message if a DSPF file is associated with a subcircuit that is not present in the design.
- `ignore`: Ignores the DSPF files associated with subcircuits that are not present in the design.
- `top`: Backannotates the DSPF files to the top-level design if they are associated with a subcircuit that is not present in the design.

sfx_ba_use_pre_layout_model

Syntax

```
sfx_ba_use_pre_layout_model = 0 | 1
```

Description

Choose the models if the pre- and post-layout netlists have different models. If set to `0`, the model is chosen from the dpf file. If set to `1`, the model is chosen from pre-layout. The default value is `0`.

sfx_ccap_cut

Syntax

`sfx_ccap_cut=`*`value`*

Description

Defines the threshold value of capacitors that need not be cut during partitioning. This option only works for the modes with partitioning enabled. The default value is 1e-13F.

sfx_ccap_esv

Syntax

`sfx_ccap_esv=`*`value`*

Description

Defines the voltage threshold of the event sensitivity related to capacitors. If not set, the value is controlled by the option `sfx_speed`. If you set this option, it overrides the predefined values specified with the option `sfx_speed`.

sfx_compress_waveform

Spectre FX provides options to control waveform compression to generate a reasonably sized waveform file for a large number of signals to be saved. The high-level option `sfx_compress_waveform` controls the tolerance of voltage and current, as shown below.

`sfx_compress_waveform`	0	1	2 (Default)	3
`sfx_compress_waveform_vabstol`		3e-4	1e-3	5e-3
`sfx_compress_waveform_vreltol`		0	0	0
`sfx_compress_waveform_iabstol`		1e-12	1e-12	1e-12
`sfx_compress_waveform_ireltol`		0.005	0.01	0.02
	Disable compression for debugging purpose	Accurate compression	Meet the general usage	A little more aggressive setting to reduce the size further

sfx_compress_waveform_vabstol

Specifies the absolute voltage error tolerance for waveform compression.

sfx_compress_waveform_vreltol

Specifies the relative voltage error tolerance for waveform compression.

sfx_compress_waveform_iabstol

Specifies the absolute current error tolerance for waveform compression.

sfx_compress_waveform_ireltol

Specifies the relative current error tolerance for waveform compression.

sfx_cshunt

Syntax

`sfx_cshunt=`*`value`*

Description

Specifies the value of grounded capacitors to be inserted for each node. For Spectre compatibility, option `cmin` is supported and is equivalent to `sfx_cshunt`.

sfx_cut_floating_gate

Syntax

```
sfx_cut_floating_gate=0 | 1
```

Description

If set to 1, Spectre FX adjusts the partition algorithm to perform the cut for the floating gate of the MOSFET. The default value is 0.

sfx_dump_floating_gate_node

Syntax

```
sfx_dump_floating_gate_node=0 | 1
```

Description

If set to `1`, Spectre FX dumps all the detected floating gate nodes to a separate file with suffix `floating_gate_node`.

If set to `0`, the tool dumps only the first 10 floating gate nodes in the log file. This is the default value.

sfx_fast_mosfet_cap

Syntax

```
sfx_fast_mosfet_cap=0 | 1
```

Description

Specifies whether to enable the fast calculation for MOSFET capacitances. When set to 1, provides better performance with some accuracy loss. It is recommended to use it only for cases that are not sensitive to nonlinear capacitances. Default value is 0, which means fast calculation is not enabled.

sfx_floating_gate_gshunt

Syntax

```
sfx_floating_gate_gshunt = value
```

Description

Specifies the value of `gshunt` placed on the floating gate nodes. The default value is 0.

Example

```
.option sfx_floating_gate_gshunt= 1e-06
```

sfx_floating_node_gshunt

Syntax

```
sfx_floating_node_gshunt = value
```

Description

Specifies the value of `gshunt` placed on the floating nodes without a conducting path to both VDD and Ground.

The default value is 0.

sfx_integ_method

Syntax

```
sfx_integ_method= gear | gear2 | traponly | euler
```

Description

Defines the integration method for transient simulation. Possible values are `gear`, `gear2`, `traponly`, and `euler`.

The default value is `gear`.

sfx_lprobe_format

Syntax

```
sfx_lprobe_format= none |digital_prefix
```

Description

Specifies the location where you want to save the logic waveforms. The default value is none, it means the logic waveforms will be stored in the original hierarchy location. If set to `digital_prefix`, Spectre FX generates a separate hierarchy started with `digital.` to store the logic waveforms for lprobe statements.

sfx_maxstep_window

Syntax

```
sfx_maxstep_window=[time1 val1 time2 val2 …]
```

Description

Sets different maxsteps for specific duration of the transient simulation. For example, apply val1 at transient time time1, val2 at time2.

Example

```
.option sfx_maxstep_window=[0 1e-8 5u 1e-9]
```

sfx_optimize

Syntax

```
.option sfx_optimize=value
```

Description

Defines the parasitic optimization in Spectre FX. The most accurate parasitic optimization can be obtained using the value `ax`. The highest parasitic optimization can be obtained using the value `vx`. Possible values are `ax`, `mx`, `lx`, and `vx`.

Example

```
.option sfx_optimize=lx
```

sfx_optimize_power

Syntax

`sfx_optimize_power=`*`value`*

Description

Defines the parasitic optimization only for the power nets that are detected automatically by Spectre FX. Possible values are `ax`, `mx`, `lx`, and `vx`.

sfx_optimize_signal

Syntax

`sfx_optimize_signal=`*`value`*

Description

Defines the parasitic optimization only for the signal nodes that are detected automatically by Spectre FX. Possible values are `ax`, `mx`, `lx`, and `vx`.

sfx_oscillator_node

Syntax

`sfx_oscillator_node=`*`[nodename]`*

Description

Specifies one node of an oscillator loop. Simulator automatically starts oscillation when the oscillator is enabled. Now the option only supports ring-type of oscillator.

Examples

```
.option sfx_oscillator_node=x1.xosc1.node1
.option sfx_oscillator_node=[x1.xosc1.node1 x1.xosc2.node1]
```

sfx_preset

Syntax

```
.option sfx_preset=value
```

Description

`sfx_preset` is a high-level simulator option that allows you to control the tradeoff between the simulation accuracy and performance. Possible values are `baseax`, `basemx`, `baselx`, `basevx`, `ax`, `mx`, `lx`, and `vx`. The value `baseax` provides the highest accuracy and the value `vx` provides the highest performance.

The `baseax`, `basemx`, `baselx`, and `basevx` values disable partitioning and provide SPICE-level accuracy while the `ax`, `mx`, `lx`, and `vx` values enable circuit partitioning and provide high performance, especially for large circuits.

The following table shows the effect of different values of the `sfx_preset` option on partitioning, speed, and optimization:

sfx_preset	baseax	basemx	baselx	basevx	ax	mx	lx	vx
partitioning	no	no	no	no	yes	yes	yes	yes
speed	ax	mx	lx	vx	ax	mx	lx	vx
optimization	ax	mx	lx	vx	ax	mx	lx	vx

Example

```
.option sfx_preset=lx
.option sfx_preset=ax subckt=pll
```

sfx_progress_percentage

Syntax

```
.option sfx_progress_percentage=val
```

Description

Controls the frequency at which the information is printed in the log file. The value is specified as a percentage.

Example

```
.option sfx_progress_percentage=10
```

The above statement states that the log file should be updated after every 10 percent of the transient simulation is completed.

sfx_rcr

Syntax

`sfx_rcr = value`

Description

Defines RC reduction, which is controlled by `sfx_optimize`, if no definition is given. The possible values are from 0 to 4. If `sfx_rcr=0`, RC reduction is disabled, which is used only for debugging purpose and the simulation may slow down significantly.

sfx_rmin

Syntax

```
sfx_rmin = value
```

Description

Resistors with values less than the specified value are shorted. The default value is 0.01 ohms.

sfx_rmax

Syntax

```
sfx_rmax = value
```

Description

Resistors with values greater than the specified value are open. The default value is 1e+9 ohms.

sfx_separate_current_waveform

Syntax

```
sfx_separate_current_waveform = 0 | 1
```

Description

Specifies whether to save a separate waveform file for current signals. By default, Spectre FX saves both voltage and current signals to the same waveform file. If `sfx_separate_current_waveform` is set to `1`, the current signals, including device current and port current, are saved to a separate waveform file. This helps in reducing the file size when many signal are probed.

sfx_speed

Syntax

```
.option sfx_speed=value
```

Description

Defines the simulation tolerance for voltage and current calculation. Possible values are `ax`, `mx`, `lx`, and `vx`. The default value is determined by the preset mode.

Example

```
.option sfx_speed=mx
```

sfx_speed_signal

Syntax

```
.option sfx_speed_signal=value
```

Description

Defines the simulation tolerance for voltage and current calculation only for the signal nodes that are detected automatically by Spectre FX. Possible values are `ax`, `mx`, `lx`, and `vx`. The setting overwrites option `sfx_speed` for signal nodes only.

Example

```
.option sfx_speed_signal=mx
```

sfx_speed_power

Syntax

```
.option sfx_speed_power=value
```

Description

Defines the simulation tolerance for voltage and current calculation only for the power nets that are detected automatically by Spectre FX. Possible values are `ax`, `mx`, `lx`, and `vx`. The setting overwrites option `sfx_speed` for power nets only.

Example

```
.option sfx_speed_power=mx
```

sfx_speed_window

Syntax

```
sfx_speed_window=[time1 val1 time2 val2 …]
```

Description

Sets different speeds for specific periods of the transient simulation. For example, apply `val1` at transient time `time1` and `val2` at `time2`.

Example

```
.option sfx_speed_window=[0 1x 5u ax]
```

sfx_support_nqs

Syntax

```
sfx_support_nqs = 0 | 1
```

Description

Supports `trnqsmod` of bsim4 and bsim3v3, if it is defined as `1`. The default value is `0`.

sfx_time_step_errtol_factor

Syntax

```
sfx_time_step_errtol_factor = value
```

Description

Specifies the factor to tune the tolerance of time step.

A value greater than 1.0 is loosening and value less than 1.0 is tightening. The default value is 1.0.

Example

```
.option sfx_time_step_errtol_factor = 0.1
```

sfx_unit_time

Syntax

`sfx_unit_time = value`

Description

Specifies the transient simulation time unit. The default value is `1e-12 (1ps)` for preset ax, mx, lx, vx, and `1e-13 (0.1ps)` for preset baseax, basemx, baselx, basevx.

Example

```
.option sfx_unit_time = 1e-15
```

sfx_waveform_flush_percentage

Syntax

```
sfx_waveform_flush_percentage = value
```

Description

Defines the percentage of the transient simulation to dump the waveforms to file.

sfx_waveform_flush_minute

Syntax

```
sfx_waveform_flush_minute = value
```

Description

Defines the specified period of transient simulation to dump the waveforms to file.

For example, if option `sfx_waveform_flush_minute` is set to `10`, the waveform data will flush every 10 minutes during transient simulation.

skip

Skips the simulation of the specified subcircuit elements. Currently, only the value `cut` is supported, which leaves the subcircuit ports disconnected. The loading effect of the skipped blocks is discarded. Possible values are `none` and `cut`.

Wildcard is supported in the parameters `subckt` and `inst`. In addition, multiple values can be specified for these parameters as a vector quoted by `[]`.

Syntax

```
skip= none | cut subckt|inst=[]
```

Example

```
.option skip=cut subckt=array
```

Tells the Spectre FX simulator to skip the simulation for all elements of the `array` subcircuit.

```
.option skip=cut inst=[xtop.xbank_left xtop.xbank_right]
```

Tells the Spectre FX simulator to skip the simulation for all elements of the `xtop.xbank_left` and `xtop.xbank_right` instances.

soft_bin

Syntax

```
soft_bin=singlemodels | no | allmodels
```

Description

Defines whether to allow the MOSFETs size to exceed the specified range in the model files, as soft binned model. It can be applied to both non-binned or binned models. Possible value are:

- `singlemodels`: Soft binned model is used only on non-binned models. This is the default value.
- `no`: The size of the MOSFETs is not allowed to exceed the given range.
- `allmodels`: Soft binned model is used on both non-binned and binned models.

6

Probing and Measuring Statements

This chapter covers the following topics:

Defining the Level or Depth Rules for Probing

You can use the `probe_level` option to specify the hierarchical depth. Possible values are:

- `top`: Count the level starting from the top level.
- `wildcard`: The level or depth specified in the `.probe` statement is applied to wildcards only. The hierarchical delimiters explicitly specified in `.probe` extend the level or depth to be probed.

In Spectre FX, the default value for `probe_level` is `wildcard`.

For compatibility with other Spectre simulators so that the level or depth of hierarchical nodes is counted from the top of the netlist, you need to explicitly set `probe_level=top`.

Related Topics

.probe or.print

Defining the Measurement Output Format

You can use the `mt_format` option to control the format of the `mt0` file. Possible values are:

- `horizontal`: Keeps the original format of the `mt0` file. This is the default value.
- `vertical`: Each measure value is specified in a separate line and the format is `meas_name=`*`val`*.

Related Topics

.measure

Probing Signals in the Backannotation Flow

For information on probing signals in the backannotation flow, refer to the section Probing Signals for the Parasitic Backannotation Flow.

.probe or.print

Syntax

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

Description

In Spectre FX, `.print` is automatically converted to `.probe`. These statements are used to probe node voltages, device currents, and port currents. The statements can contain hierarchical names and wildcards for nodes, ports, or elements, and can be embedded within the scope of a subcircuit.

The statements also support the following:

- Multiple statements in the netlist file
- Output variables in different formats

Note: Spectre FX ignores non-existent signal names and prints a warning message with the names of those signals in the log file.

Arguments

`tran`	Defines the analysis type as transient. This is an optional parameter and can be skipped because Spectre FX supports only transient analysis.

`ov1 ov2....ovn`	Name of the output variable (it can be the node voltage, branch current, port current, Verilog-A instance port voltage, or Verilog-A instance internal variable value). ■ `v(`*`node_name`*`)` probes the *`node_name`* voltage. *`node_name`* can be hierarchical and can contain question marks and wildcards. For example: `v(x?1.*.n*)`. ■ `i(`*`element_name`*`)` probes the branch current output through the element *`element_name`*. *`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). ■ `x(`*`instance_port_name`*`)` returns the current flowing into the subcircuit port, including all lower hierarchical subcircuit ports. It can be used to probe the power and ground ports of an instance, even if the ports are defined as global nodes 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)`. ■ `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 and also 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`. `var_name` can be either a port name or an internal variable name of a Verilog-A module. `veriloga_instance` is the instance name of a Verilog-A 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`. `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 `1`, only the nodes at the current level are applied (default value is infinity). This parameter has an alias `level`. The option probe_level decides how to count the hierarchical depth.
`subckt=`*`name`*	Specifies the subcircuit to which 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`	Specifies the output variables to be excluded from the probe. The names can be node or element names and can contain wildcards. This parameter has an alias `except`.
`preserve`	Defines the content of the nodes probed with wildcard scoping. Possible values are: `none`: Probes all the nodes and ports connected to active devices (default). Nodes connected only to passive elements are not probed. `all`: Probes all nodes, including the nodes connected to passive elements, and probes all ports. `port`: Probes only the ports in subcircuits.

Examples

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

Tells the Spectre FX simulator to probe the voltage at node `n1` and the current `i1` for element `M1`. The voltage difference between nodes `n2` and `n3` is probed and assigned to `vdiff`. In addition, an expression of voltages at nodes `n1` and `n2` is probed and assigned to `expr1`.

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

Tells the Spectre FX simulator to probe the voltages for all the nodes in the subcircuit named `VCO` and one level below in the circuit hierarchy. The current of the resistor r1 is also probed 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 `.probe tran v(*) i(r1) depth = 2` is written in the subcircuit definition of `VCO` in the netlist file.

```
.probe tran x(xtop1.block1.in)
```

Tells the Spectre FX simulator to report the current of port `in` for instance `block1`, which is instantiated at the top-level block `xtop`.

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

Tells the Spectre FX simulator to probe the current of ports for instance `xtop` and all instances below.

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

Probes the voltages for all nodes in `VCO`, including port names and internal nodes that are only connected to resistors and capacitors.

```
.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*`.

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

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

```
.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 Verilog-A instance `IO.ANA.VREG`, as well as all the port voltages and internal variable values for Verilog-A instances that match the name `IO.ANA.C*`.

Related Topics

.lprobe or .lprint

save

.measure

.lprobe or .lprint

Syntax

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

Description

In Spectre FX, `.lprint` is automatically converted to `.lprobe`. 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.

Arguments

Name	Description
`tran`	Defines the analysis type (transient).
`ov1`, `ov2`	Specifies the simple output variables and uses the `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.
`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.
`depth = value`	Specifies the depth in the circuit hierarchy to which a wildcard name applies to. If set to `1`, only the nodes at the current level are applied (the default value is infinity).
`subckt = name`	Specifies the subcircuit to which the statement applies to. By default, it applies to the top level. If the statement is already specified 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` probes only the ports in subcircuits.

Examples

```
.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.

```
.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 to the circuit call path from the top level to `INV`. This is equivalent to a situation where the statement `'.lprobe tran v(*) depth = 2'` is in the subcircuit definition of `INV` in the netlist file.

```
.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.

```
.lprobe tran low = 0.5 high = 4.5 v(*) exclude=net* exclude=bl*
```

All node voltages are probed except for node voltages matching the pattern `net*` and `bl*`.

Related Topics

.probe or.print

save

.measure

save

Spectre FX enables you to save the signals by using the `save` statement in Spectre syntax.

Note: This simulator does not support the `oppoint` parameter in the `save` statement `save dev:oppoint`.

Related Topics

save statement

.probe or.print

.lprobe or .lprint

.measure

.measure

Syntax

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

Description

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 `.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 Spectre FX simulator supports linked measure statements applicable for all the measure functions listed below. Some measure statements may depend on others by having the 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 at the top level or in the same subcircuit definition).

Related Topics

Functions Supported by .measure in Spectre FX

save statement

.probe or.print

.lprobe or .lprint

Functions Supported by .measure in Spectre FX

The Spectre FX simulator supports the following measure function types:

- current and power
- Average, RMS, Min, Max, Peak-to-Peak, and Integral
- Find and When
- Rise, Fall, and Delay
- Parameter with Expressions

Related Topics

.measure

current and power

Description

Used for current and power analysis on elements or subcircuits.

Examples

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

Tells the Spectre FX simulator to measure the maximum current of port `out` of instance `xtop.x23` and all the instances below.

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

Tells the Spectre FX simulator to measure the maximum power of port `out` of instance `xtop.x23` and all the instances below.

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

Tells the Spectre FX simulator to measure the average power on element `r1`, from `0 ns` to `1 us`.

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

Tells the Spectre FX simulator to measure the integral power (total energy) of port `out` of instance `xtop.x23` and all the below instances.

Related Topics

.measure

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

Syntax

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

Arguments

Name	Description
tran	Specifies the transient analysis for the measurement. **Note:** The Spectre FX simulator only supports measurement of transient analysis.
meas_name	User-defined measurement name.
ov1	Name of the output variable (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=`*value*	Start time for the measurement period.
`to=`*value*	End time for the measurement period.

Examples

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

Tells the Spectre FX simulator to calculate the average voltage of node `1` from `0 ns` to `1us`, evaluating the result with variable `avg1`.

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

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

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

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

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

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

Related Topics

.measure

Find and When

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

or

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

or

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

Note: The `from`/`to` pair, `at`, and `td` arguments cannot be specified together in the same `.measure` statement.

Arguments

Name	Description
tran	Specifies the transient analysis for the measurement. **Note:** The Spectre FX simulator only supports measurement of transient analysis.
meas_name	User-defined measurement name.
when|*find*	Specifies the `when` and `find` functions.
ov1, *ov2*, *ov3*	Names of the output variables (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 the `.measure` statement.

Name	Description
`from=`*`value`*	Start time for the measurement period.
`to=`*`value`*	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
```

Related Topics

.measure

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

Name	Description
`tran`	Specifies the transient analysis for the measurement. **Note:** The Spectre FX 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 Spectre FX simulator uses to determine the propagation delay with respect to `trig_var`.
`val=`*`value`*	Value of `trig_var` or `targ_var`.
`td=`*`value`*	Specifies the time the measurement starts. The simulator counts the number of `cross`, `rise`, or `fall` events that occur after the `td` value. The default value is 0.0.

Name	Description
`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 the measurement start time. The value can be a real time or a measurement result from a previous `.measure` statement.

Examples

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

Tells the Spectre FX 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 value is 0.5 volts on the first falling edge.

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

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

Tells the Spectre FX 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.

Related Topics

.measure

Parameter with Expressions

```
.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

```
.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'
```

In the above example, the measure `avg12` returns the sum of the values from `avg1` and `avg2`.

```
.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
```

In the above example, the measure `delay1` is calculated based on the results of `time1` and `avg01`.

Related Topics

.measure

7

EMIR Analysis and Post-Layout Simulations

This section covers the following topics:

EMIR Analysis

Spectre® FX provides a powerful transistor-level EMIR solution. This dynamic power net and signal net EMIR capability uses a patented technology and is designed to provide high-capacity and high-performance EMIR analyses. The Spectre FX EMIR solution provides a set of advanced features covering dynamic and static EMIR, power gate support, and differential IR drop. The flow is fully integrated into Virtuoso® ADE Explorer and Virtuoso® ADE Assembler and the results can be post-processed with Voltus™- Fi Custom Power Integrity Solution.

Related Topics

Spectre FX EMIR Technology, Product, and Flow Overview

Getting Started with Spectre FX EMIR Analysis

Power Gate Support

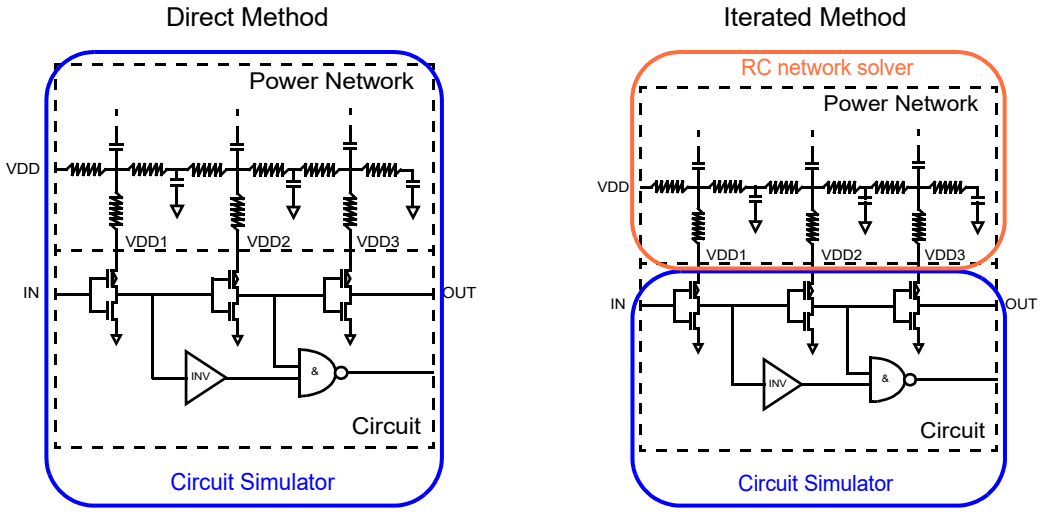
Handling the Complexity of DSPF/SPEF files

Advanced Analyses

Spectre FX EMIR Technology, Product, and Flow Overview

In an EMIR flow, a circuit is evaluated together with the parasitic resistor and capacitor network, which models the IR drop or the EM effect. There are two general approaches of solving such a problem:

- **Direct EMIR analysis -** When high accuracy is needed, a brute-force simulation of the entire system (circuit plus parasitic resistances and capacitances) can be performed to accurately calculate the EMIR of any net. The EMIR simulation performance and capacity is subject to the limitation of the circuit simulator being used.
- **Iterated EMIR analysis -** In order to conduct EMIR simulation on circuits with much larger power and signal nets within a much shorter time, an alternative is to decouple the nonlinear circuit simulation from the linear RC net analysis. User can iterate the linear RC net analysis by modifying the layout, however, the nonlinear circuit simulation is performed only once. The decoupling of the linear RC nets from the nonlinear circuit is not mathematically equivalent to the original design and certain inaccuracy is introduced, however, the user receives the benefit of simulation performance and capacity.



High accuracy, limited performance/capacity

High performance/capacity, sufficient accuracy

For high accuracy EMIR analysis of small design blocks, or designs with smaller numbers of RC nets, direct EMIR analysis is recommended.

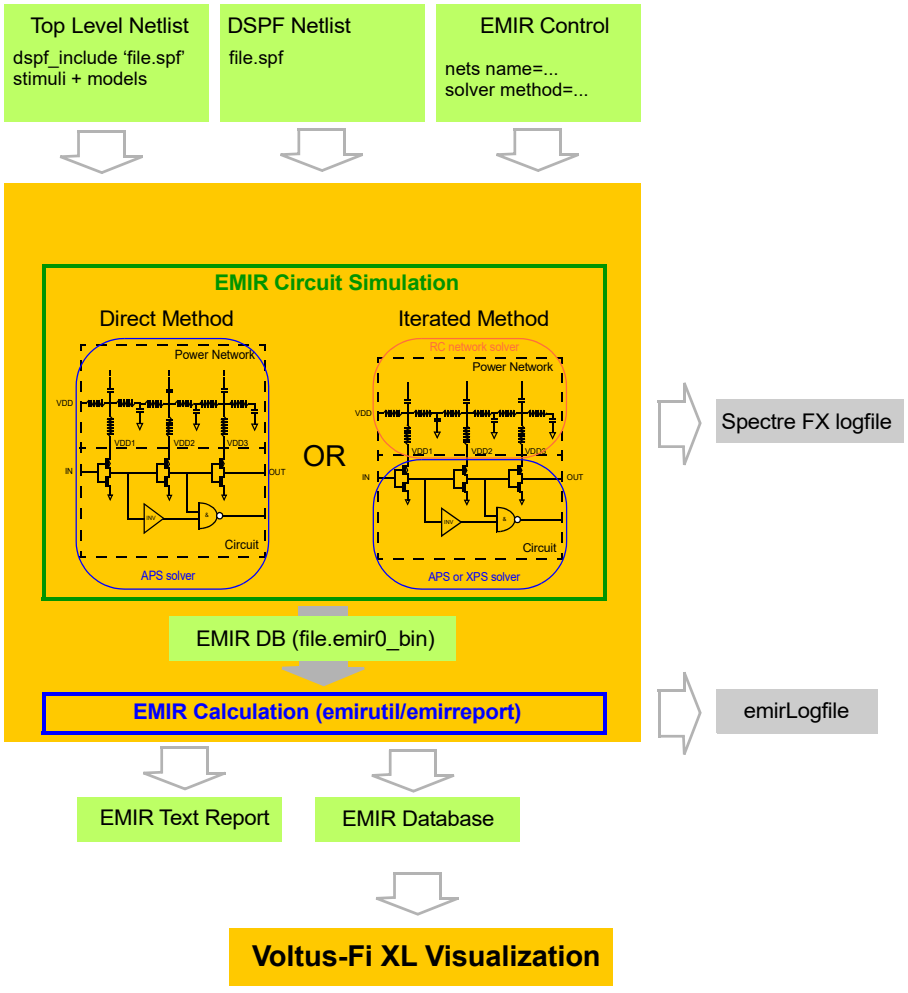
To gain higher performance and higher capacity on medium-to-large designs, the iterated EMIR analysis method is recommended.

The Spectre FX EMIR flow requires a complete testbench that contains the DSPF files (with the parasitic and instance sections describing the circuit to be analyzed) stimuli, device, and models. .

Use the `+emir` option on the Spectre FX command line to enable the EMIR analysis during circuit simulation. The details of EMIR analysis, such as the type of analyses, nets to be analyzed, output to be created, are specified using an EMIR configuration file.

Using either direct EMIR analysis or iterated EMIR analysis, the circuit simulation is first performed with various voltages and currents being calculated. Next, a standalone tool, `emirreport`/`emirutil`, is called to post-process the simulation results and generate the IR and EM reports. By default, the post-processing step is invoked automatically.

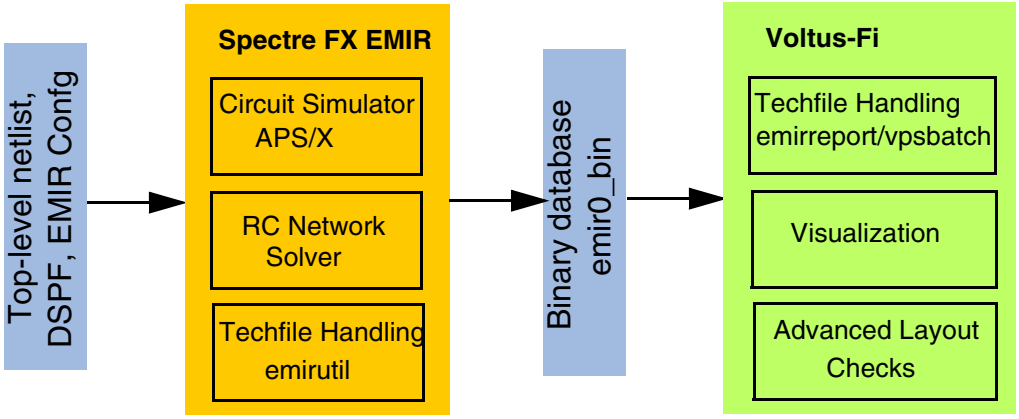
The output of the EMIR analysis is an EMIR text/html report, which lists the EM and/or IR information of all the nets requested. In addition, *Voltus-Fi XL* can be used to visualize the IR drop and EM current violations in Virtuoso Layout Editor.



Note that while the Spectre FX EMIR flow uses the DSPF representation of the designs, it is not dependent on any post-layout backannotation or stitching capability. The flow also covers designs with power gates.

Product Overview and EMIR Flow

The EMIR product consists of the Spectre FX EMIR simulation and the *Voltus-Fi XL* visualization. The circuit and the RC network simulation are handled by Spectre FX EMIR. The result is stored in a binary database which contains the average, rms, max IR drop, and EM values. *Voltus-Fi XL* reads the information from the binary database, evaluates whether the currents are above the limits defined in the technology file, and visualizes the results in the layout.

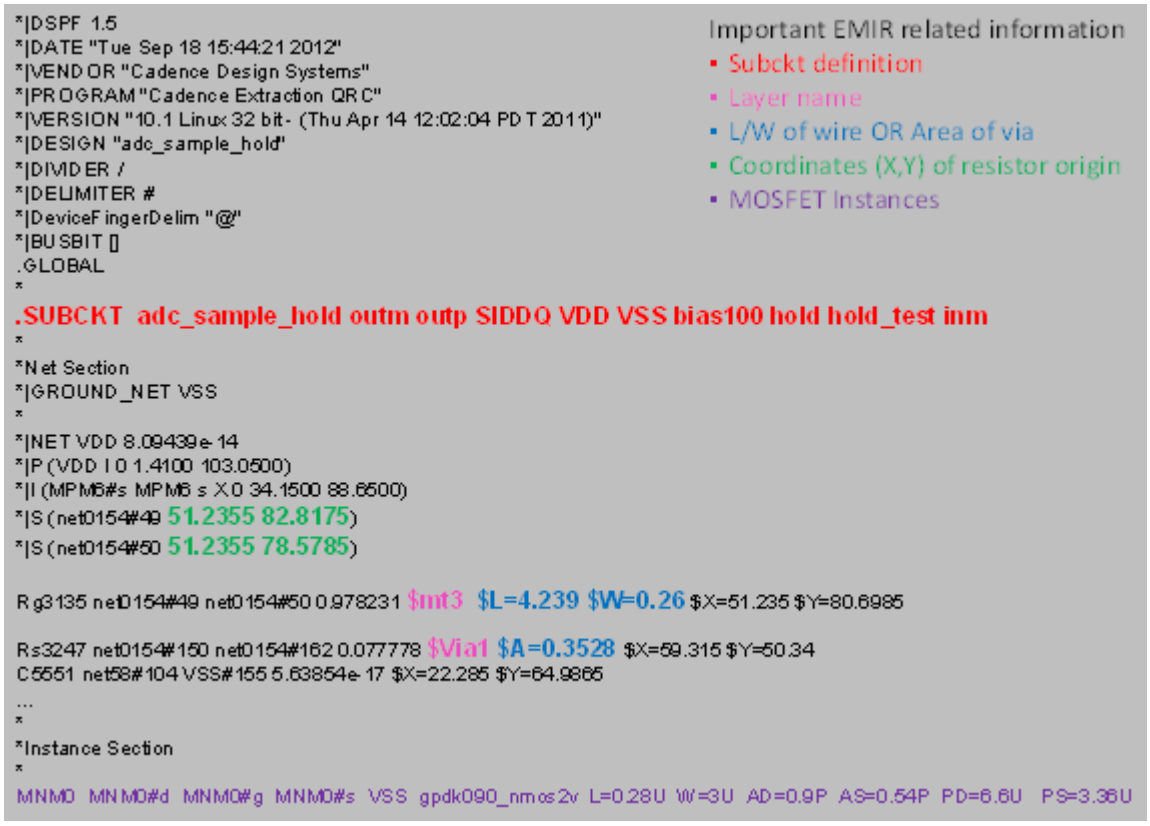


Different utilities are responsible for creating the IR drop and EM text reports. For advanced node designs for which the technology file format is ICT or `qrctechfile`, the utility `emirreport` (or `vpsbatch`) is used to generate the text report. For older technology nodes which use `emdatafile` format, the utility `emirutil` is used to create the text reports.

DSPF requirements

The Spectre FX EMIR flow is based on a DSPF representation of the analyzed design. The DSPF file is required to contain all electrical and geometric information needed for EMIR

analysis. The following figure shows a representative DSPF netlist with all important information for EMIR being marked in color.



EM Rule Support

EM rules are defined in DRM, and part of the technology files. The *Voltus-Fi XL*– Spectre FX EMIR solution supports the following technology file formats.

- ICT, ICT-EM (EM rules only)
- `qrctechfile`
- `emdatafile`

Advanced node technologies require special EM rules that are only supported in ICT/ICT-EM and `qrctechfile` format. Refer to the *Quantus QRC Techgen Reference Manual* for details.

`emdatafile` format is used for legacy technology nodes. In future, it is expected to be replaced by ICT and `qrctechfile` format.

Various Cadence EMIR tools support only subsets of these techfile formats (for example, Voltus does not support ICT file, but supports `ICT_em`). `ICT_em` is the only format supported by all Cadence EMIR technologies, Voltus, EAD, and Voltus-Fi L and XL.

Getting Started with Spectre FX EMIR Analysis

To perform EMIR analysis, first create a complete simulation testbench with the DSPF files containing the postlayout data of the design. Specify the fingered devices in the `instance` section and the parasitic resistors and capacitors in the `net` section. Use the `dspf_include` statement to read the DSPF content, as shown below.

```
dspf_include "sram.spf"                          (Spectre syntax)

.dspf_include "sram.spf"                         (SPICE syntax)
```

`dspf_include` provides special features like port order adjustments, or handling of duplicated subcircuits, which are not available in `include`/`.include`. Therefore, do not use `include`/`.include` in the EMIR flow for including the DSPF file.

Large DSPF files may be split into multiple files, for example, `sram.dspf`, `sram.dspf.1`, `sram.dspf.2`, and so on. If the top-level DSPF file (`sram.dspf`) is included with the `dspf_include` statement, Spectre FX searches for any related file (`sram.dspf.1`, `sram.dspf.2`, ...) and read them automatically.

Setting up a correct postlayout simulation test bench is vital to successful EMIR analysis (see Handling the Complexity of DSPF/SPEF files).

Once the testbench is set up, you can perform a regular (non-EMIR) postlayout simulation with the `spectrefx` command to ensure that the testbench contains no error, and the circuit behavior is as expected.

```
% spectrefx input.sp
```

To perform the same simulation with EMIR analysis added, you need to create and include an EMIR control file with the `spectrefx` command, as shown below.

```
% spectrefx +emir=emir.conf  input.scs
```

The EMIR analyses is enabled by using the `+emir` Spectre FX command-line option. All EMIR-related commands are defined in the EMIR control file.

EMIR Control File

EMIR analysis is performed based on the first transient analysis statement in the netlist. An EMIR control file contains all of the settings and options that are relevant to an EMIR analysis. Some of the options that can be specified using the EMIR control file are: the names of the

nets to be analyzed, the type of analyses to be performed, the EMIR analysis time window, choice of direct or iterated methods, settings for RC simulation, and location of EM rule file.

The following sample control file describes EMIR analysis based on the `X1` instance. A maximum IR drop analysis is performed on nets `VDD` and `VSS`, while RMS-based EM analysis is performed on all nets inside the `X1` instance. The two-stage iterated method is used, and the current density limits for the EM report are referenced from file `.emfile.txt`. Only one subcircuit instance is allowed for EMIR analysis for a Spectre FX run.

Example EMIR control file (emir.config)

```
net name=[X1.VDD X1.VSS] analysis=[vmax iavg] net name=[X1.*] analysis=[irms]
solver method=iterated
emirutil techfile="./emfile.txt"
```

The EMIR control file supports the Spectre line continuation character + and comment-line characters * and //.

The following table summarizes the supported EMIR control file options:

The following table summarizes the supported EMIR control file options:

Keyword	Option Set	Explanation	Default Value
net	`name=[instance1.net1 instance1.net2....]`	Defines the nets for which the analysis is performed. `instance` defines the instance of the subcircuit containing the net. `net` defines the net name inside the subcircuit instance as it is defined in the `DSPF *| NET` definition. If the DSPF file is included at the top level, the instance name is not required. Wildcards aresupported for net names but not instance names.	`none`
	`analysis=[imax iavg iavgpos iavgneg iavgabs irms vmax vavg]`	`peakEM`, `avgEM`, `avgEM` for i>0, `avgEM` for i<0, `avgabsEM`, `rmsEM`, `peakIR`, `avgIR`. `imax`, `iavg`, `iavgabs`, `irms`, and `vmax` data are always written to the binary database independent of user settings. **Note:** If *Voltus-Fi* `acpeak` analysis is required, then `imax` and `irms` analyses need to be enabled in Spectre FX EMIR. `imax` means `max(abs(current waveform))`	`iavg, vmax`

	`pwrgate=[I1.VDD I1.VDD_INT...]global=[I1.VDD]`	Enables power gate handling. You need to specify the power supply net driving the power gate (vsource connected) and the internal power supply net driven by the powergate. If one power supply net drives multiple power gates, you need to specify one statement with all internal power supply nets. If none of the powergate nets is driven by vsource, then the `global` statement can be used to define the global net.	`none`

	`analysis=[sigvmax sigvavg] reftype=[avg max min pin] [findsrc=yes|no][refnode=name][vref=value]`	`analysis=sigvmax` - max signal net IR drop. `analysis=sigvavg` - avg signal net IR drop. `reftype=avg` - reference voltage for IR drop is the average voltage of all subnodes at every time point. This is the default. `reftype=max` - reference voltage for IR drop is the maximum voltage of all subnodes at every time point. `reftype=min` - reference voltage for IR drop is the minimum voltage of all subnodes at every time point. `reftype=pin findsrc=yes` - Reference voltage of IR drop is taken from vsource, which is traced back from `*P|` node. `reftype=pin findsrc=no` - Reference voltage of IR drop is taken from `*P|` node. `refnode` - use the reference voltage from the specified reference node. `vref` - use the voltage value as reference and report the value as vmax and not as sigvmax in the report.	`none`

	`name [instance.net1...] r2r_refnet=VSS analysis=[r2rvmin] [r2rvmin_layer=mt1]r2rvmin_refnet_layer=[m2] r2rvmin_tap_inst=[MN1* MN2*] r2rvmin_refnet_tap_inst=[MN3* MN4*] r2rvmin_model=[pch_...] r2rvmin_refnet_model=[nch_...] r2rvmin_bbox=[1.0 1.0 269.5 279.5] r2rvmin_refnet_bbox=[...] r2rvmin_node_report=[on|off] r2rvmin_rule_name=name r2rvmin_excl_layer=[m6] r2rvmin_refnet_excl_layer=[m9] r2rvmin_excl_tap_inst=[MN5] r2rvmin_refnet_excl_tap_inst=[MN6] r2rvmin_excl_model=[pch...] r2rvmin_refnet_excl_model=[pch...] r2rvmin_excl_bbox=[32.99 95.92 41.01 99.659] r2rvmin_refnet_excl_bbox=[...]`	Worst case differential (rail-to-rail) IR drop between thespecified power supply node and the reference net written to the `*.rpt.r2r` file. If r2r_refnet is not specified, the ground node is automatically detected. At each time point, the sub node with the highest IR drop is selected for power and ground net and the differential voltage is calculated. The worst case differential IR drop over the EMIR time window is reported and the worst case power and ground waveforms are written to the waveform file. Optional scoping (combined with AND) includes layer names, tap device names, and tap device model names for both power supply node and refnet. A boundary box can be defined for power supply net and refnet. Optionally, a name can be assigned to each `r2rvmin` option. `r2rvmin_node_report` enables additional report of nodes and XY coordinates (default is off). Exclude statements supported for layers, tap device instances, models, and bounding boxes. All exclude statements allow you to specify file content. For example, `r2rvmin_refnet_excl_tap_inst_file=refexcl.txt`.	`none`
	`vref=value`	Defines the optional voltage reference value for `vmax` and `vavg` analyses.	`none`

	`exclude=[netname1 netname2...]`	Excludes the specified nets when `name=[*]` is used.	`none`
solver	`method=[auto|direct|iterated|profonly|iteronly | iterated_split]`	`auto` - direct for Spectre FX and Spectre X direct - forced direct method `iterated` - forced iterated method `profonly` - iterated flow - circuit profile generation `iteronly` - iterated flow - RC network iteration (requires `solver inputwf=iterated.emirtap.pwl`) `iterated_split` - iterated flow - Same as iterated; however, it runs the first and second stages as separate processes to reduce peak memory usage	`auto`
	`global_pwr_net=[name1 name2 …]`	By default, IR drop is enabled ony for DSPF subcircuit ports that are driven by a DC vsource. If a subcircuit port is not connected to a DC vsource, it can be defined as a power supply node using the `global_pwr_net` option and IR drop on this net can be calculated by specifying the `analysis=vmax` statement on this net. The IR drop reference value used is the maximum value of the voltage waveform at the specified port. This feature is only supported in iterated method. Wildcarding is not allowed. Once enabled, the net can be used in the `pwrgate` statement.	`none`

	`tstep=value`	Change the solver to a fixed timestep specified by *`value`*. The unit is in seconds.	`none`
	`min_tstep=value`	Minimum time step for adaptive time step. The unit is in seconds.	`1ps`
	`rcsolve_multi_proc=[on|off]`	If set to `on`, enables parallel (forked) processes for second stage network solving. You can use the `+mt` command-line option to specify the maximum number of cores to use. Small-sized nets (< `rcsolve_1t_netsize`) are solved first with eight forked single-thread processes. Medium-sized nets are solved next with two forked four-thread processes. Large nets (>`rcsolve_4t_netsize`) are solved with one main process using eight threads.	`off`
	`ignore_tap=[name1 name2...]`	Ignores the specified tap devices in dynamic, static EMIR, and SPGS. The specified tap devices are changed to subnodes and do not carry any current. In addition, they are not reported. It applies to only the second stage of the iterated method and is not supported in the direct method. Wildcards are supported.	`none`
	`report_tapc=[off|text]`	Report the net and tap device capacitance values for nets being defined in the UTI command. The capacitance report is written to a file with extension `rpt_tapc`.	`off`

	`inputwf=file_name`	Specifies the waveform file that contains the results from circuit simulation when `method=iteronly` is used.	`none`
`static`	`ifile=`*`filename`*	Enables static EMIR analysis. The filename defines the subcircuit instance port currents. Static EMIR analysis cannot be combined with dynamic EMIR analysis.	`none`
	`exclude=[net=[name1 name2] dev=[name1 name2]]`	Exclude the tap devices from static current distribution. A 0A current is assigned to the specified tap devices. Wildcarding is supported for both `exclude` and `dev`. For example: `exclude=[net=[I1.VDD] dev=[MPM3*]].`	`none`
	`report_tapi=[true|false]`	Enables the printing of individual tap device currents in static EMIR analysis. The current values are written to a file with the extension `rpt_tapi`.	`false`

emirutil	techfile="emDataFile"	Specifies the EM rule file in the emdatafile, qrctechfile, or ICT file format. The EM rule file can be predefined in the configuration file using the EMTECHFILE environment variable, as shown below. setenv EMTECHFILE="<*path*>/*emdatafile*" It can also be defined in the .cdsinit file as follows: envSetVal("spectre.envOpts" "emTechFile" string <*path to the file*> Case-insensitive layer names are supported for easier match between the DPSF file and emData file. The rule file defined in the configuration file has higher priority followed by the rule file specified in .cdsinit. The rule file specified using the environment variable has the least priority. Alternatively, you can use the emdatafile="*emDataFile*" option to specify the techfile in the emdata format, qrctechfile="*qrcTechFile*" to specify the QRC tech files, and ictfile="ictfile" to specify the file in ICT format.	none

	`layermapfile="mapfile"`	Optional file that defines the mapping between the layer names in the DSPF file and the layer names in the technology file.	`none`
	`EMOnlyICTFile="emOnlyICTfile"`	Specifies the file containing only the EM rules. It is usually used in combination with qrcTechFile.	`none`
	`autorun=[true|false]`	`true`: run `emirutil` automatically after the solver to generate a text report. `false`: manually run `emirutil`.	`true`
	`report=[text|layout | html]`	Defines the type of report created by `emirutil`. `text`: creates only the text report. `layout`: creates only the layout violation map. `html`: creates the report in html format. **Note:** Multiple entries are supported.	`text`
	`notation=[s|e]`	Notation for the text and html reports. `e`: engineering scale number (for example, 5.02m) `s`: scientific notation (for example, 5.02e-3)	`e`
	`sort_by_net=yes|no`	Report IR and EM results per net, or all nets. `yes`: report EMIR results per net. `no`: combine EMIR results of all nets into one report.	`yes`

time	`window=[start1 stop1 start2 stop2 ...]`	Time window to which the EMIR analysis is applied. It applies to all nets for which the analysis is defined. Multiple and overlapping windows are supported. An individual report is created for each time window. You can use different time windows for emirutil v/s Spectre simulation.	`[0 tend]`

The EMIR control file supports the Spectre line continuation character `+`, and the comment lines start with character `*` or string `//`.

Checking the Progress of EMIR Analysis

As the simulation runs, Spectre FX sends messages to the screen and the simulation log file that show the progress of the simulation and provide statistical information. Following information may be significant to your EMIR analysis when you use the iterated method.

Hardware configuration and run-time machine loading

In the beginning of a simulation session, Spectre FX prints the hardware configuration (physical memory, CPU core specification) and the run-time machine status (available memory, CPU core operating frequency, CPU loading). Since EMIR analysis often involves large-scale designs with gigabytes of data, it is important to ensure that there is sufficient memory available, with CPU operating at full speed (not in power saving mode), and the machine loading is light.

```
User: user1 Host: lnx-user1 HostID: CD0A1190   PID: 26141
Memory  available: 10.2268 GB  physical: 16.6236 GB
CPU Type:        Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz
          Processor PhysicalID CoreID Frequency Load
              0         0        0     3192.5     0.3
              1         0        1     3192.5     0.2
              2         0        2     3192.5     0.3
              …
```

Reading the EMIR Control File

Spectre FX reads the EMIR control file before processing the circuit being simulated. All valid EMIR options are reported in the following section:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~
EMIR Analysis Configuration
~~~~~~~~~~~~~~~~~~~~~~~~~~

nets             name=[i1.VDD i1.VSS] analysis=[vmax iavg]
nets             name=[i1.*] analysis=[irms]
solver           method=iterated
emirutil         techfile = emDataFile.txt
```

If one of the specified EMIR options is not displayed in the summary, check the related warnings, and correct the syntax.

Running the Circuit Simulation to Produce Voltage Profiles

At this point, with the simplification of power and signal nets, Spectre FX is ready to simulate the nonlinear circuit and produce voltage profiles needed for iterated EMIR analyses. This step is similar to a regular circuit simulation, where Spectre FX first parses the circuit, reports circuit inventory, initiates a DC analysis, and finishes the step with a transient analysis.

```
Creating probes for EMIR analysis: 57 voltage probes, 0 current probes

Time for setting up EMIR tap devices: CPU = 0 s (0h 0m 0s), elapsed = 0 s (0h 0m 0s).


Time for setting up EMIR analysis: CPU = 580.0 ms (0h 0m 0s), elapsed = 80.0 ms (0h
0m 0s).
…
**************************************************
Transient Analysis `transient1': time = (0 s -> 20 ns)
**************************************************
…
......9......8......7......6......5......4......3......2......1.0
Number of accepted tran steps =918

Total time required for tran analysis `tran': CPU = 4.88835 s,
elapsed = 625.093 ms, util. = 782%.

Time accumulated: CPU = 7.64419 s, elapsed = 5.33127 s.
Peak resident memory used = 158 Mbytes.


Time for creating waveform database: CPU = 10.0 ms (0h 0m 0s), elapsed = 10.0 ms
(0h 0m 0s).
```

The reported transient time and the accumulated time are important measures to be considered when optimizing performance. The voltage profiles of the tap devices (active devices connecting to the power and signal RC nets) are stored in the pwl files, and used as input for the second stage EMIR simulation.

This step of EMIR simulation is fully multi-threaded. It uses the number of cores defined by the `+mt` command-line option.

Simulating the Parasitic RC Nets

In the second stage of the iterated EMIR analysis, the power and signal nets are simulated together with the tap devices. During the simulation, the tap device terminal voltages over time is taken from the voltage profiles created in the first stage. In the second stage, nets are solved individually, except for coupled nets, which are solved together.

The RC network solver writes all IR drop voltage and EM current values to the binary database with the extension `emir0_bin`. Important log file content is written to the `EMIR PARASITICS ANNOTATION SUMMARY` section, which provides information on the nets and R/C elements processed. In addition, the number of solved time steps per net, and the elapsed time for the RC solving are essential when optimizing performance.

```
*****************
Start EMIR RC Analysis at Sat May  6 00:37:07 2023
*****************

Current resident memory used = 153.27 Mbytes.


Reading EMIR configuration file:  input.raw/input.emirtap.conf

~~~~~~~~~~~~~~~~~~~~~~~~~~
EMIR Analysis Configuration
~~~~~~~~~~~~~~~~~~~~~~~~~~
solver          method = iteronly
solver          inputwf = input.raw/input.emirtap/input.emirtap.pwl

Current resident memory used = 153.34 Mbytes, Peak memory used = 153.34 Mbytes, at
Sat May  6 00:37:07 2023
emir 2nd stage solver pre_sim time is set to 0n
Time for Pre process           : CPU = 0 s (0h 0m 0s), elapsed = 0 s (0h 0m 0s).

…


Maximum grounded vsource value = 2.5v
…

Searching and processing cross coupling capacitors in SPF file
./adc_sample_hold.dspf
Sat May  6 00:37:07 2023


Cross coupling capacitors found and processed: 0

EMIR RC analysis cross coupling capacitor parsing elapsed time: 0.020 s.
…
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
EMIR PARASITICS ANNOTATION SUMMARY
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Nets:                       parsed            18 processed            18
Cross coupling capacitors: parsed             0 processed             0
Capacitors:                 parsed          3709 processed          3709
Resistors:                  parsed          6803 processed          6712
Nodes:                      parsed          5955 processed          5955

Nets annotated with C-only:        0  Nets annotated with RC:       18
No errors and No warnings(fixed errors)


Postprocessing 2nd stage waveform files Sat May  6 00:37:09 2023
Done at Sat May  6 00:37:09 2023

…


~~~~~~~~~~~~~~~~~
EMIR NET COVERAGE
~~~~~~~~~~~~~~~~~
Number of non-emir nets = 0
Number of emir nets     = 18
          Number of skipped nets             = 0
          Number of successfully solved nets = 18
          Number of failed nets              = 0
Number of accepted time steps = 4570 (average 253.889 per net)
Number of total time steps = 4570
Number of rejected time steps = 0 (average 0 per net)

average number of Newton Iterations per dc/tran solve = 0.0784656
average number of solver iterations = 3.3846
EMIR PERFORMANCE INFO:
          Net               Elapsed Time         Time Steps
          i1.sample         0.26s                813
          i1.hold           0.25s                780
          i1.Vcm            0.24s                685
          i1.Vcp            0.24s                685
          i1.inp            0.18s                559

Finished EMIR RC Analysis at Sat May  6 00:37:09 2023


Total time required for EMIR RC analysis: CPU = 4.1  s (0h 0m 4s), elapsed = 1.6
s (0h 0m 1s).
Peak resident memory used = 158.2 Mbytes
```

This step of the EMIR simulation is fully multi-threaded. It uses the number of cores defined by the `+mt` command-line option.

After the EMIR data is available in the binary database, another step is performed to create the text reports for IR drop, EM currents, pin currents, and power gate information. The

following output shows the related section in the log file (some content is only written to stdout):

```
Creating EMIR report, check 'input.raw/input.emirtap.emirlog' for more
information.

Pre process           : CPU = 0 s (0h 0m 0s), elapsed = 0 s (0h 0m 0s)
Database building     : CPU = 120.0 ms (0h 0m 0s), elapsed = 150.0 ms (0h 0m 0s)
Simulating nets       : CPU = 4.0  s (0h 0m 4s), elapsed = 1.4  s (0h 0m 1s)
Circuit update      : CPU = 290.0 ms (0h 0m 0s), elapsed = 130.0 ms (0h 0m 0s)
RC solver           : CPU = 2.1  s (0h 0m 2s), elapsed = 860.0 ms (0h 0m 0s)
Time step prediction: CPU = 300.0 ms (0h 0m 0s), elapsed = 170.0 ms (0h 0m 0s)
Device evaluation   : CPU = 820.0 ms (0h 0m 0s), elapsed = 180.0 ms (0h 0m 0s)
Output              : CPU = 470.0 ms (0h 0m 0s), elapsed = 130.0 ms (0h 0m 0s)
Instance section      : CPU = 0 s (0h 0m 0s), elapsed = 10.0 ms (0h 0m 0s)
Post process          : CPU = 0 s (0h 0m 0s), elapsed = 0 s (0h 0m 0s)
Generate text report  : CPU = 120.0 ms (0h 0m 0s), elapsed = 140.0 ms (0h 0m 0s)
```

At the end of the simulation, Spectre FX reports the total simulation time and peak memory used.

```
Aggregate audit (3:53:45 AM, Tue Jan 12, 2016):
Time used: CPU = 2.38 s, elapsed = 63.3 s (1m  3.3s), util. = 3.76%.
Time spent in licensing: elapsed = 60.1 s (1m  0.1s), percentage of total = 94.8%.
Peak memory used = 68.8 Mbytes.
```

Viewing EMIR Results

Spectre FX EMIR writes the IR voltage and EM current values to a binary database with the extension `emir0_bin`. In addition, it automatically calls the `emirreport` or `emirutil` binary for creating the text or html reports.

The textual/html IR drop report (file extension: `rpt_ir/rpt_ir.html`) lists the voltage drop per net (in the order of largest to smallest), and provides information about the time the maximum IR drop occurred, as well as the layer information and layer coordinates. An example section of an IR drop report is shown below.

```
VOLTAGE DROP RESULTS

…

-------------------- "VDD" NET: Vref = 2.5V ----------------------------------
max - drop

VOLTAGE-DROP    NETNAME         TIME            LAYER           X              Y
(V)                            (s)                            (um)           (um)
6.212m          MPM1@16:s       20.781p         mwires          75.120         76.080
6.212m          MPM3@9:s        20.781p         cont            75.120         76.080
6.211m          VDD:567         20.781p         metal1          75.120         95.840
```

The EM report (file extension: `rpt_em/rpt_em.html`) lists the pass/fail results for each resistor segment of a net, and provides information about the resistor name, layer, current, layer width, density (current divided by width), density limit, number of vias required, and the coordinates.

```
ELECTROMIGRATION ANALYSIS RESULTS

…

---------------------- NET "VDD" -------------------------------
avg

There is no resistor whose current density exceeds the limit.

%failed     resistor layer   current  width    pathLength density     limit needed
width/#vias  X1       Y1       X2       Y2     resistance
(A)    (um)     (um)       (A/um)       (A/um)       (um/#)            (um)    (um)
(um)     (um)     (ohm)
pass-16.05%  rr1027   via2    1.504m   4.480    155.735    335.783u     400.000u
14(16)              60.190  206.060 60.190  206.060 0.0875
pass-30.86%  rr1026   via2    309.737u 1.120    155.735    276.551u     400.000u
3(4)                60.200  199.280 60.200  199.280 0.350
pass-36.10%  rr1019   via2    1.145m   4.480    155.735    255.598u     400.000u
11(16)
```

The EMIR text and html reports can also be created using an existing EMIR binary database using the following command:

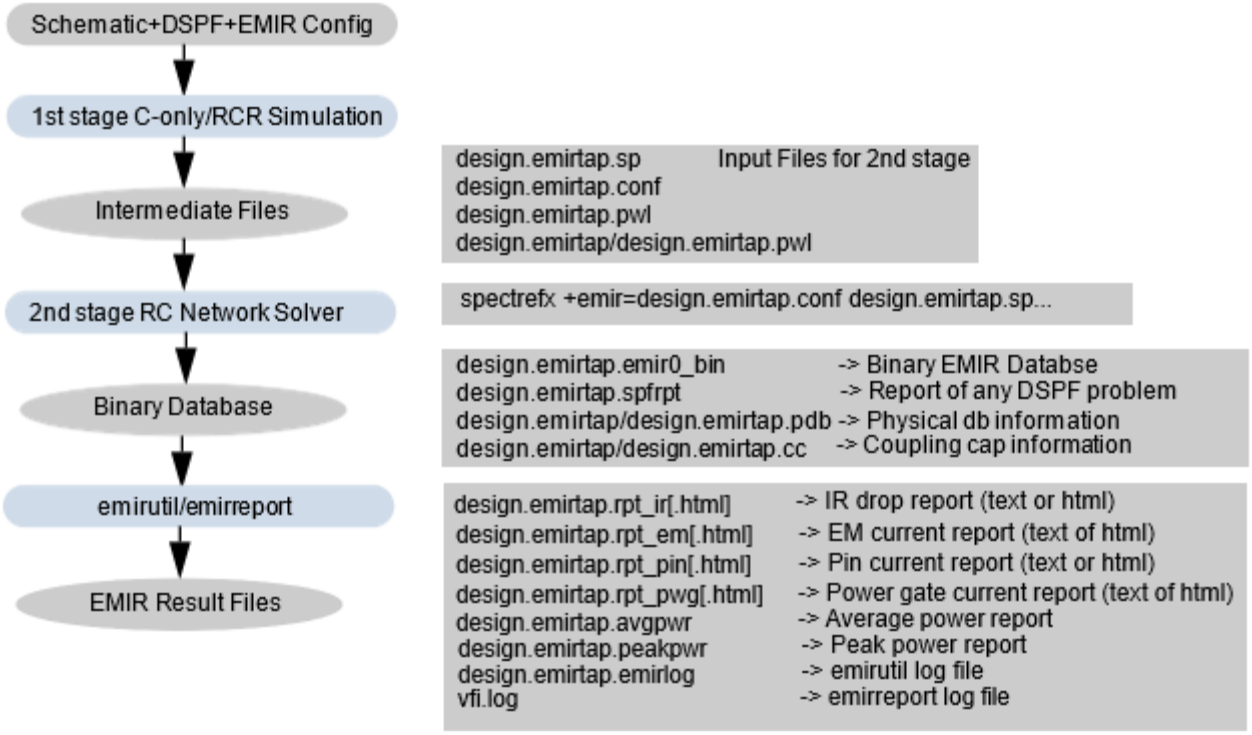
```
$ emirutil -db input.emir0_bin -control emir.conf
```

For more information on `emirutil`/`vpsbatch` utility for creating the reports, and for the visualization of the EMIR analysis results in the layout, refer to the *Voltus-Fi* user manual.

If there are multiple `emir0_bin` files created by the same Spectre FX EMIR simulation, it is sufficient to specify only the first binary database file with the `-db` option. The utility automatically identifies all other binary database files created by the Spectre FX EMIR simulation.

Data Flow

The data flow of the iterated EMIR flow is shown in the following flowchart. The first stage EMIR circuit simulation creates the tap device voltage profiles which are stored in the pwl files, and used in the second stage. The second stage RC network solver writes all IR drop and EM current values to a binary database with the extension `emir0_bin`. This binary database is used by `emirutil`/`emirreport` for creating the text/html reports, and by *Voltus-Fi XL* to visualize the results in the layout.



All intermediate and output files are written into the Spectre FX output (raw) directory. Some files shown in the figure are only created when the related feature is enabled in the EMIR config file.

If multiple EMIR windows are used in the same EMIR simulation, a set of files is created for each EMIR time window. For example:

```
Time window 1: emir0_bin, rpt_ir, rpt_em, …
Time window 2: emir1_bin, rpt_ir1, rpt_em1, …
```

The data flow for the direct EMIR method is the same except that the intermediate files are not created. Only the iterated method EMIR files have the `emirtap` file extension.

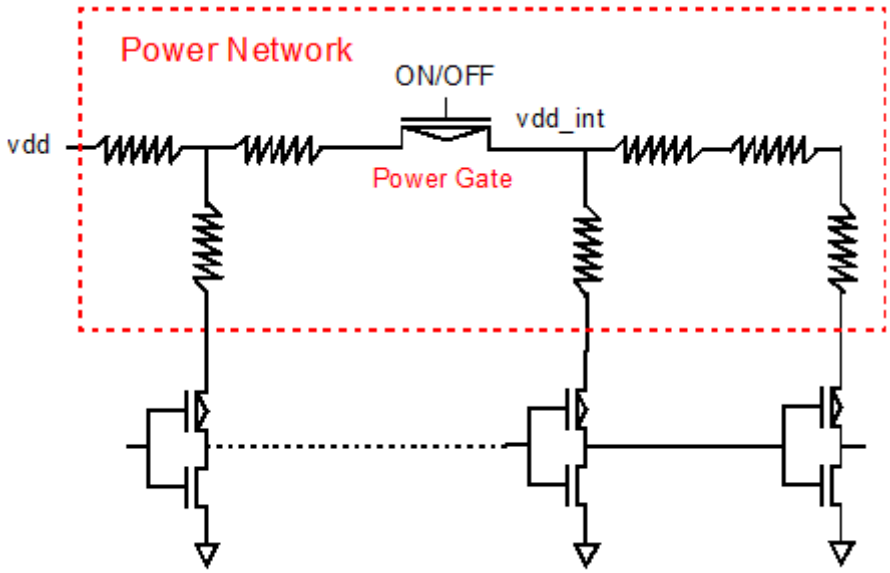
Direct method: `design.emir0_bin`, `design.rpt_ir`, `design.rpt_em`

Iterated method: `design.emirtap.emir0_bin`, `design.emirtap.rpt_ir`, `design.emirtap.rpt_em`

For large designs, the intermediate (`pwl`, `pdb`) and EMIR database files (`emir0_bin`) can be compressed by using the `eisopt zipfiles=2` option in the EMIR config file.

Power Gate Support

Power networks may contain power gates (four-terminal MOSFET device) which enable or disable the power supply in the circuit. These power gates split the power supply RC network into two parts; the RC network driving the power gate, and the RC network being driven by the power gate. These nets are strongly coupled together and they should be simulated together in EMIR analysis.



To invoke power gating handling, add the power gate setting in the EMIR control file (`emir.config`), as shown below.

```
net pwrgate=[vdd vdd_int] analysis=[vmax]
```

Both, the global power supply net `vdd` driving the power gates, and the virtual internal power supply net `vdd_int` driven by the power gates, need to be specified in the `net` statement in any order. Wildcards are supported for internal power supply nets, but not for power supply nets. If one power supply net drives multiple power gates, one statement with all power supply nets needs to be defined, as shown below.

```
net pwrgate=[vdd vdd_int1 vdd_int2] analysis=[vmax]
```

The IR drop report includes both the power supply net in the usual report file name, for example, `input.rpt_ir` and the internal power supply net is reported in a file, such as `input.rpt_pwg` or `input.emirtap.rpt_pwg` for direct and iterated methods respectively. The IR drop for the virtual power net includes the voltage drop on the global power net, the voltage over the power gate, and the voltage drop on the virtual power net. EM analysis is performed as usual, if specified.

In addition, an important parameter `Ton` is also reported in the `.rpt_pwg` file. It reports the time taken to power-up the terminal of the internal power supply net to 95% of the power supply level (VDD).

Note: The parameter `Ton` can be infinity if the internal power supply level does not reach 95% of VDD.

Each `pwrgate` statement allows you to define only one global power node and the related virtual power nodes. If a design contains multiple global nodes with power gates and virtual nodes, then, for each of them, a separate `pwrgate` statement is required.

Serial power gate structures are supported. For such gate structures, not only the top-level power supply net and the internal supply net, but also the net between the serial power gates needs to be specified.

Handling the Complexity of DSPF/SPEF files

SPF Checker

The DSPF files are created with parasitic extraction tools like QRC. The content and format of a DSPF file is heavily dependent on the extraction tool and the settings. Often, simulation problems occur due to problems in the DSPF file.

SPF Checker is a utility that analyses a DSPF file, reports problems which may cause simulation problems, and creates an EMIR configuration file with the recommended mapping statements for EMIR analysis.

The SPF checker utility can be located at `<`*`spectre_install_dir`*`>/tools.<plat>/bin/ spfchecker`.

The SPF checker utility can be run as follows::

```
spfchecker SPF_FILE_NAME[-detail <value>] [-message <value>] [-log LOG_FILE_NAME]
     -c EMIR_CONF_NAME] [-ckt=subckt_name] [-ctl=filename] [+lorder licenseList]
     -force -help
```

Argument	Description
`-dspf`	If the input is in DSPF format (default).
`-spef`	If the input is in SPEF format.
`-outdir <dir_name>`	Set the output directory for the checker results. Default is `.`

Argument	Description
`-detail`	Report detailed information (statistics) for NETs whose subnodes are more than the value specified using `-d`. Default is 0.
`-message`	Set the maximum number of messages for each type of error. Default is 50.
`-log name`	Set the checker log file name. If this argument is not specified, a file with the name `SPF_FILE_NAME.chklog` is created by default.
`-c name`	Create the EMIR configuration file. If this argument is not specified, a file with the name `SPF_FILE_NAME.emir_conf` is created by default.
`-opt name`	Create a backannotation option file. If this argument is not specified, a file with the name `SPF_FILE_NAME.stitch_opt` is created by default.
`-ckt subckt`	Define the subcircuit scope. Default is the first subcircuit definition found.
`+lorder`	Specify the license checkout order. The default checkout order is `PRODUCT:MMSIM`
`-fix name`	Fix the common connection errors and create a new DSPF file.
`-dump_cc name`	Dump the cross coupling capacitors into the named file
`-dump_lvl level`	Set the level to dump the original coupling capacitors. Default is 0.
`-force`	Overwrite the previous SPF checker results.
`-ctl file`	Exclude the parasitic resistors specified in the list. For example, ctl file: spf_ignore_layers = [M1 M2]
`-help`	Display help related to the SPF Checker utility.

The SPF checker creates the following files:

- `test.spf.chklog` - Detailed SPF checker log file
- `test.spf.spfinfo` - DSPF element inventory
- `test.spf.emir_conf` - Recommended settings for EMIR config file

- `test.spf.stitch_opts` - Recommended (SPICE format) settings for DSPF backannotation

Before running any EMIR analyses, ensure that your DSPF file is clean. This can be achieved by running the `spfchecker` utility and by confirming that `spfchecker` does not report an error message in the `*chklog` file. Typical error messages in the `chklog` file are:

```
SPF-0003:    2           ERROR: NETs broken into pieces
SPF-0016:    6           ERROR:Instances with different connection from
                                the NET Section
SPF-0019:    430         ERROR: NETs with sub-nodes not connecting to any
                                parasitics
SPF-0028:     1           ERROR: Instance model terminal definition conflicts
SPF-0039:     3854807     ERROR: Layer is extracted as both via and metal layer
```

Simulating a DSPF file for which `spfchecker` reported errors may generate invalid EMIR simulation results. If `spfchecker` reports warnings and no errors, the DSPF file can be used in EMIR analysis.

Additionally, `spfchecker` prepares content for the EMIR config file into the `*.emir_conf` file. This content can either manually be copied into the EMIR config file, or included with an `include` statement:

```
include test.spf.emir_conf
```

Port Order Handling

When setting up the EMIR simulation, you need to ensure that the subcircuit port order in the DSPF file matches the port order of the instance call in the top-level netlist. The `.dspf_include` command provides advanced functionality for port order mapping, as given below.

```
.dpsf_include file.spf  port_order=[spf|sch]
```

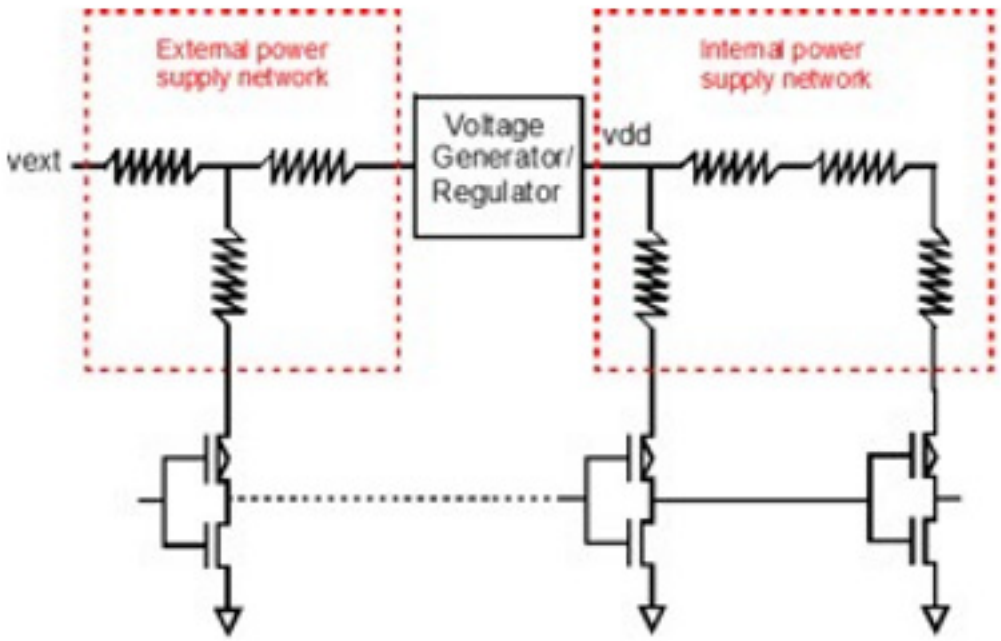
- `port_order=sch`: (Default). The port order is taken from schematic subcircuit definition. The same port number and names are required. If the schematic subcircuit definition is not available, a war:ning is issued in the log file, and DSPF port order is used.
- `port_order=spf`: The port order is taken from the DSPF subcircuit definition.

Advanced Analyses

Signal Net IR Drop Analysis

IR drop analysis (analysis=[vmax vavg]) is typically applied to power nets that are driven by voltage sources, which provides a constant reference voltage over the EMIR window. However, designs may have internal generators or regulators. For the generator driven nets, an IR drop analysis may be important. These nets behave like signal nets since their voltage value changes over time.

The Spectre FX EMIR signal net IR drop analysis (analysis=[sigvmax sigvavg]) allows you to perform IR drop analysis on such nets. However, you need to define the reference voltage being used for the IR drop calculation. The recommended setting for generators is refnode=vdd. At each time point, it calculates the voltage at the reference node vdd, and calculates the IR drop in reference to this voltage.




```
net name=[i1.vdd] analysis=[sigvmax] refnode=vdd
```

The following are supported for defining the reference voltage:

- `reftype=max` - Use the maximum voltage of all nodes at each time point.
- `reftype=avg` - Use the average voltage over all nodes at each time point (default).

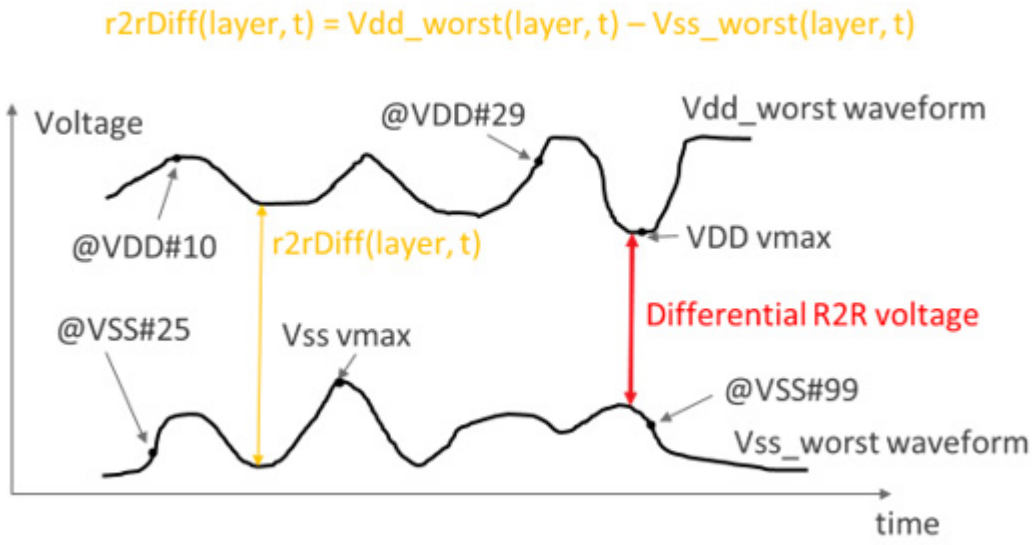
- `reftype=pin findsrc=true` - For the given nets, use the vsource connected to `*|P` node.
- `reftype=pin findsrc=false` - For the given nets, use the `*|P` node as the reference, and not vsource.
- `refnode=`*`name`* - Use the specified sub node of `*|NET` as reference.
- `vref=`*`value`* - Use the specified voltage as reference.

When using `reftype=max` for a net with three subnodes: `net1:1 (1V)`, `net1:2 (2V)`, and `net1:3 (3V)`, the detected reference voltage will be 3V, and the IR drop calculated will be 2V for `net1:1`, 1V for `net1:2`, and 0V for `net1:3`. When using `reftype=avg` the reference voltage will be 2V, and the IR drop calculated will be 1V for `net1:1`, 0V for `net1:2`, and -1V for `net1:3`.

The signal net IR drop analysis is automatically changed to a regular IR drop analysis (`vmax`), if `vref=value` is specified, or if the net (during the EMIR time window) is driven by a constant voltage source.

Differential (Rail-to-Rail) IR Drop Analysis

The differential IR drop feature allows you to identify the worst-case IR drop between the specified power supply and the ground node. At each time point, it calculates the worst IR drop over all the power supply sub nodes, and the worst IR drop over all the ground sub nodes. At each time point, it also calculates and reports the worst case rail-to-rail voltage for the given EMIR time window.



The differential IR drop analysis is enabled with the `analysis=r2rvmin` option. Multiple power supply nodes can be specified. If the ground node is not specified in the `r2r_refnet` statement, Spectre FX automatically detects the global ground node.

```
net name=[VDD] r2r_refnet=VSS analysis=[r2rvmin]
```

The following is an example of a differential IR drop report:

```
**r2rvmin RESULT
Power_net_name R2rvmin_value  R2rvmin@time   Attribute      Ref_net_name
Ideal_r2rvmin  Ref_Attribute  Rule_name
                (V)                                                    (V)
VDD           2.489         20.781p       0              VSS           2.500
1               _R2RVMIN_AUTO_RULE_1
```

The `r2rvmin` feature allows you to apply the analysis only to selected layers, selected tap devices, tap devices of a user-defined model, and selected bounding boxes. Combinations are supported and combined with the AND behavior. Excluding tap nodes based on their layer, model, or instance information is also supported. To differentiate between multiple `r2rvmin` reports, you can apply the rule names to each `r2rvmin` statement (`r2rvmin_rule_name`). Refer to the `r2rvmin` option in the table under the EMIR Control File section for more information. The differential (rail-to-rail) IR drop analysis is only supported in the direct method.

Static EMIR Analysis

Static EMIR analysis enables you to evaluate IR drop and EM currents based on the specified current consumptions for subcircuit instances without running a transient or DC simulation. The specified currents are distributed to the tap devices based on the width and length ratios of devices in the design. The IR drop and EM current analysis is performed based on the current at each tap device.

To enable static EMIR analysis, you need to use the `static ifile` option in the EMIR configuration file, as shown below.

```
net name=[I1.VDD I1.VSS] analysis=[vavg iavg]
static ifile="static_currents.txt"
```

For static EMIR analysis, `vmax` and `vavg` for IR drop and `imax`, `irms`, and `iavg` values for EM currents remain the same, and therefore, just one can be selected in the statement.

The subcircuit instance currents are defined (in A) in the `static_currents.txt` file with the subcircuit instance name, subcircuit port name, and the current flowing in the port. This current is distributed to all MOSFET tap devices. No current distribution is done for non-

MOSFET tap devices. In that case, these devices are excluded from the block-level based current distribution. The following is an example of the file:

```
I1 VDD 0.001
I1.I2 VDD 0.005
I1.I2/I3 VDD 0.00001
I1.I2.I3/I4 VDD 0.000005
I1 VSS -0.005
I1.MPM1@19 VDD 0                //exclude tap device from current assignment
I1.MPM3@15 VDD 0.0001           //define tap device current,exclude from current
                                  assignment
x0.QQ0 VSS 0.1 b               //define current for non-MOSFET device and terminal
```

The accuracy of static EMIR analysis strongly depends on the detailed current information provided in `static_ifile`. Therefore, it is highly recommended to provide the current consumption for every subcircuit instance.

The results of static EMIR analysis are written to the same IR drop and EM current text reports, and into the same EM binary database as dynamic EMIR analysis.

Note: Only one of static or dynamic EMIR analysis can be performed from the same simulation session, and not both. When you run an static EMIR analysis, all other Spectre FX analyses, such as `dc` and `tran` are ignored.

Static EMIR analysis can also be applied to designs with power gates. For this, you need to specify the `pwrgate` option, as follows:

```
net name=[I1.VDD] analysis=[vavg iavg]
static ifile="static_currents.txt"
net pwrgate=[I1.VDD I1.VDD_INT] analysis=[vavg iavg]
```

You need to specify the power gate current in the `static_currents.txt` file, as shown below.

```
I1 VDD 0.001
I1 VDD_INT 0.0005
```

The static EMIR current file for the top-level EMIR subcircuit can be generated while running a dynamic EMIR analysis by using the `output=power` option in the time window statement.

Spectre FX EMIR Analysis Using Voltus-XFi

Voltus-XFi Custom Power Integrity Solution is used for transistor-level power and signal integrity analysis, which includes multi-mode simulation for EM and IR analysis. This solution combines Cadence® Quantus™ RC extraction, Spectre FX EMIR simulation, and Voltus-XFi visualization in the Cadence Virtuoso® platform. This combination of products simplifies the use model of the flow by using a common setup for all the tools. You do not have to run the flow in separate steps.

Post-layout Simulation Methodologies

There are three post-layout simulation methodologies:

- Flat RC Netlist File

  A flat netlist containing many elements and devices. One example is to directly include the DSPF file. Another example is the netlist from the Quantus QRC extracted view, which is often used in Virtuoso® ADE Explorer or Virtuoso® ADE Assembler.

- Hierarchical RC Netlist File

  A netlist file that contains a hierarchy of extracted subcircuits.

- Backannotation of Parasitic Files

  DSPF/SPF/DPF files containing parasitic information. This methodology combines the parasitic information with the pre-layout netlist through backannotation. It enables Spectre FX to automatically plug in the parasitic elements during the simulation.

  Note: The SPEF format is currently not supported in this release.

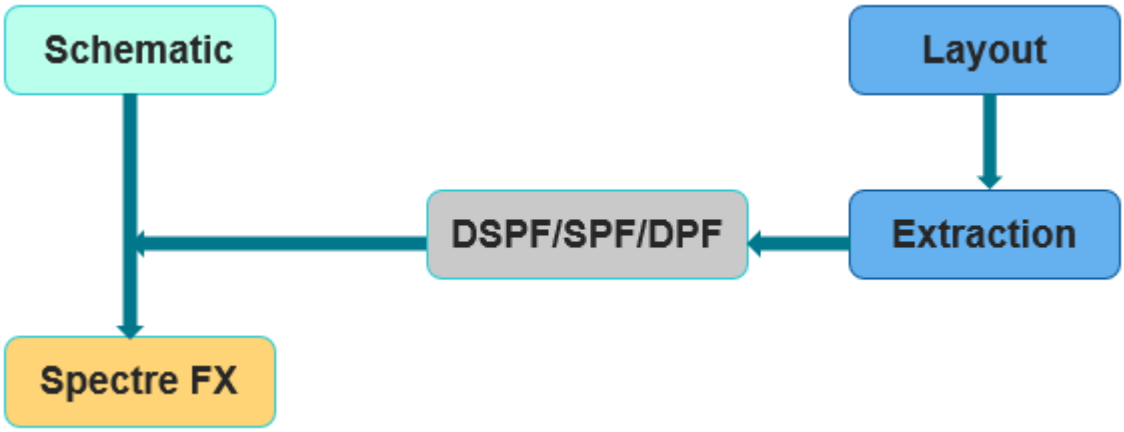
Related Topics

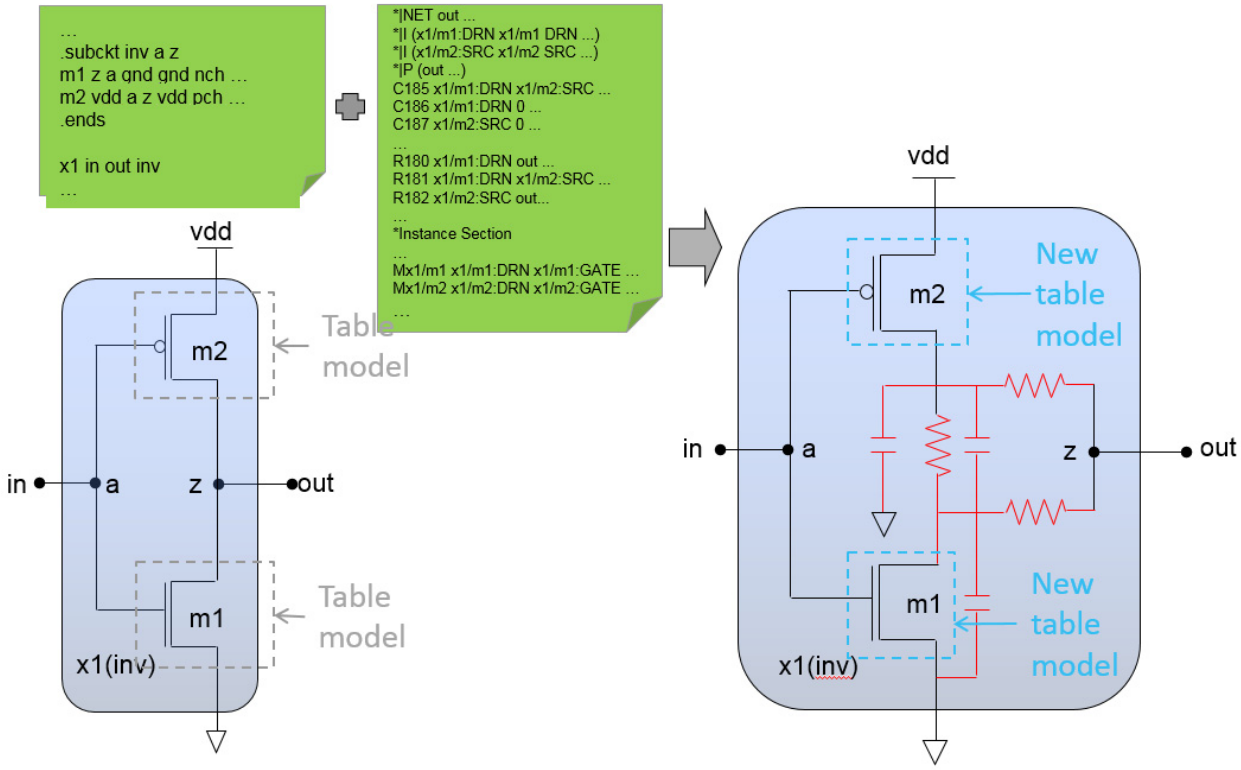
The Parasitic Backannotation Flow

Control Options for the Parasitic Backannotation Flow

The Parasitic Backannotation Flow

The following figure shows the parasitic backannotation flow:





The simulation database is built upon the pre-layout netlist, as shown at the left of the figure. The parasitics from the parasitic files (DSPF/SPF/DPF) are backannotated into the database, as shown at the right. The hierarchy and the net names from the pre-layout netlist are

retained, and the device models are regenerated according to the device parasitic information (DPF/Instance Section from the DSPF). The nodes are backannotated with R and C from the DSPF/SPF Net section.

Related Topics

Post-layout Simulation Methodologies

Control Options for the Parasitic Backannotation Flow

Control Options for the Parasitic Backannotation Flow

The control options for the parasitic backannotation flow can be grouped under the following four categories:

- Parasitic File Loading Options
- Parsing Options
- Selective Backannotation Options
- RC Handling Options

The following table describes the options in these categories:

Option	Description
Parasitic File Loading Options	
sfx_ba_dspf	Specifies the DSPF file with both `RC NET` section and `Instance` section to be read.
sfx_ba_dspf_net	Specifies the DSPF file or SPF file to be read for the RC NET section.
sfx_ba_dspf_instance	Specifies the DSPF file or DPF file to be read for the Instance section.
sfx_ba_unrecognized_subckt_action	Specifies how to handle a DPSF file whose subcircuit name does not match any in pre-layout.
sfx_ba_use_pre_layout_model	Choose the models if the pre-layout and post-layout netlists have different models.
RC Handling Options	
sfx_ba_rmin	Shorts the resistor during the backannotation stage. The default value is 0.1 ohm.
sfx_ba_ccap_min	Splits and grounds the coupling capacitors whose value is less than the specified value during the backannotation stage. The default value is 0 F.
Parsing Options	
sfx_ba_instance_scale	Scales the geometry of devices in the Instance section of the device, which can be different from the scaling in pre-layout.

sfx_ba_remove_prefix	Specifies the characters of an instance to be removed so that the post-layout element can match the one in the pre-layout.
sfx_ba_hier_delimiter	Defines the hierarchical delimiter in the DSPF file.
sfx_ba_bus_delimiter	Defines the bus delimiter in the DSPF file.
sfx_ba_finger_delimiter_string	Defines the finger delimiter string for instances in the DSPF file.
sfx_ba_ground_dangling_ccap	Defines how to handle coupling capacitors with an invalid node connection.
sfx_ba_rep_pin	Specifies how to choose the representative node for an RC NET.
sfx_ba_probe_node	Specifies how to probe the nodes in an RC NET.
sfx_ba_probe_node_name	Specifies how to specify the signal name if you set the `sfx_ba_probe_node` option to a value other than `rep` because the backannotated nodes need to be probed.
Selective Backannotation Options	
sfx_ba_skip_net	Specifies the nets that are to be skipped for backannotation.
sfx_ba_skip_net_file	Specifies a text file that contains the list of nets to be skipped for backannotation.
sfx_ba_gcap_net	Backannotates the total ground capacitance of the specified nets.
sfx_ba_gcap_net_file	Specifies a text file that contains the list of nets whose ground capacitances need to be backannotated.
sfx_ba_rc_net	Backannotates the RC tree of the specified nets as defined in the RC NET section.
sfx_ba_rc_net_file	Specifies a text file that contains the list of nets whose RC tree needs to be backannotated.
sfx_ba_ccap_net	Backannotates the coupling and ground capacitances of the specified nets.
sfx_ba_ccap_net_file	Specifies a text file that contains the list of nets whose total coupling and ground capacitance needs to be backannotated.

sfx_ba_gcap_net_cth	Backannotates the total ground capacitance of the net whose total capacitance is smaller than the specified value.

Important

Unlike traditional simulation options, wherein if the same option is specified multiple times, the last one takes precedence, most backannotation options, when specified multiple times, accumulate the assigned values.

Related Topics

Post-layout Simulation Methodologies

The Parasitic Backannotation Flow

Guidelines for Parasitic Backannotation Options

Parasitic Backannotation Report

Guidelines for Parasitic Backannotation Options

For the selective backannotation options, consider the following guidelines to define `net` in the RC section:

- Wildcards are supported.
- For multiple net names in the option definition, specify them within `[]`.
- If you specify the selective backannotation options multiple times, all specified settings are considered and combined.

  Note: The above is also applicable to a few other backannotation options, such as `sfx_ba_dspf_net` and `sfx_ba_dspf_instance` options, but not all. For example, in case of the `sfx_ba_instance_scale` option, only the last specified value is considered.
- The hierarchical delimiter is from the DSPF file (because `net` exists in the DSPF file). The mapping of `net` in the options is matched with the DSPF file RC NET and not directly with the pre-layout nodes.
- If you are specifying `net` in files for options `sfx_ba_skip_net_file`, `sfx_ba_gcap_net_file`, and so on, ensure that each `net` is specified in a separate line. Multiple `net` entries in one line are not supported.

If a `net` is specified using by multiple selective backannotation statements, the precedence is defined by the following two rules:

- **Rule 1**: The net name without a wildcard overwrites the names matched by wildcards in all the other statements. For signals with wildcard, their priority is the same.
- **Rule 2**: The precedence of options is: `sfx_ba_skip_net` > `sfx_ba_rc_net` > `sfx_ba_ccap_net` > `sfx_ba_gcap_net`.

Related Topics

Post-layout Simulation Methodologies

The Parasitic Backannotation Flow

Control Options for the Parasitic Backannotation Flow

Probing Signals for the Parasitic Backannotation Flow

Probing Signals for the Parasitic Backannotation Flow

Spectre FX supports probing of subnodes and instance pins on an RC net. It is not recommended to do it for all RC nets in a big DSPF file because it may cause a huge number of signals to be saved and a big waveform file to be generated.

To enable this feature, you need to set the following two backannotation options

- sfx_ba_probe_node: Specifies how to probe the nodes in an RC NET that can be set based on RC NET with wildcards supported. Possible values are `rep`, `pin`, `all`, `gate`, and `primary_gate`. The default value is `rep`.
- sfx_ba_probe_node_name: Specifies how to specify the signal name. Possible values are `brief` and `full`. The default value is `brief`.

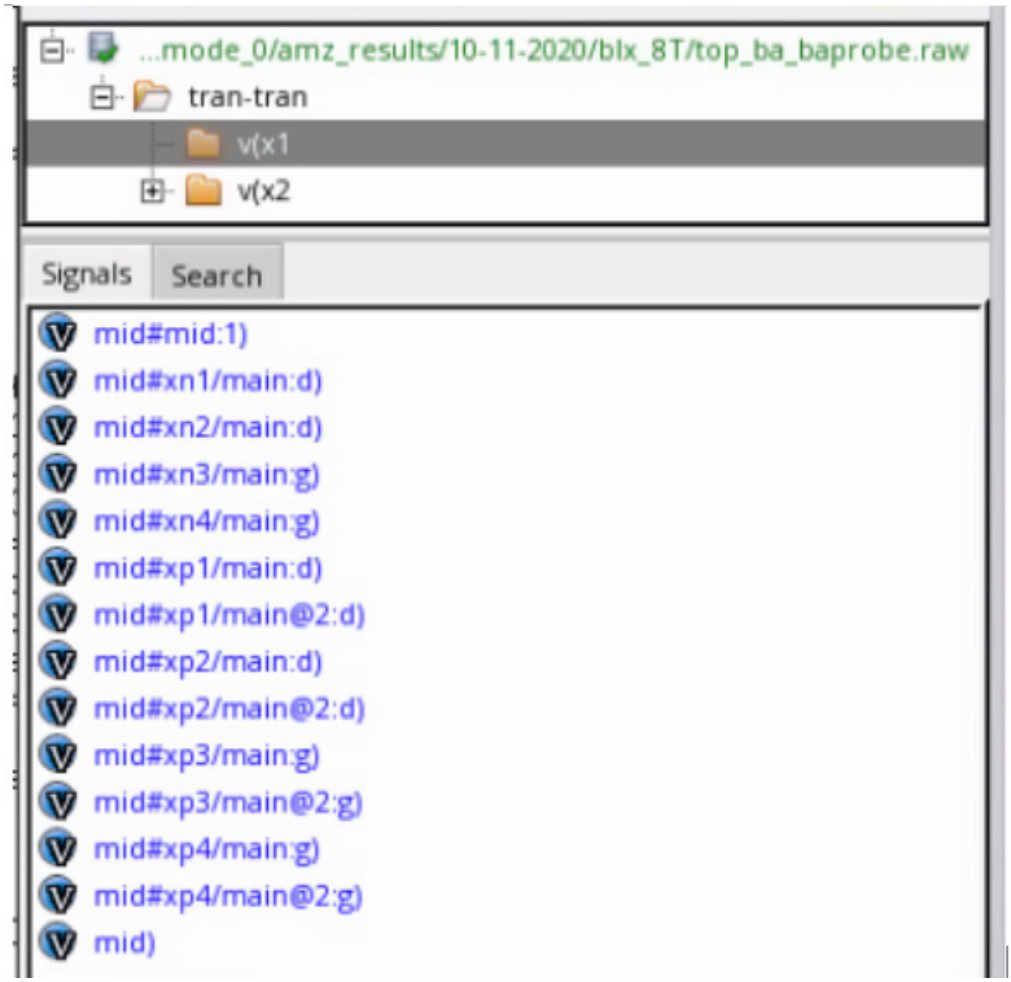
Wildcards are supported in both the `.probe` statement and the `sfx_ba_probe_node` option. You can limit the scope of the final signals saved using either method.

The following example probes all the nodes in the RC Net section of the DSPF file and display the full name for them.

```
.probe v(x1.mid)
.option sfx_ba_probe_node=all
.option sfx_ba_probe_node_name=full
```

Here, the pre-layout signal `v(x1.mid)` is specified, therefore, the probed signals are limited to it.

Finally, all the nodes in RC NET corresponding to the pre-layout node `x1.mid` are probed. The signal list is shown below.



Related Topics

Post-layout Simulation Methodologies

The Parasitic Backannotation Flow

Control Options for the Parasitic Backannotation Flow

Parasitic Backannotation Report

The Spectre FX log file displays the statistical data and status update related to parasitic file during parsing and annotation.

This can be explained with the following examples:

Example 1: Multiple Small SPF/DPF files

In the netlist, the SPF/DPF files are invoked using the following options:

```
.option sfx_ba_dspf_net=test.spf subckt=[inv_2x]
.option sfx_ba_dspf_instance=test.dpf subckt=[inv_2x]
.option sfx_ba_dspf_net=test_ver2.spf subckt=[inv_2x_ver2]
.option sfx_ba_dspf_instance=test_ver2.dpf subckt=[inv_2x_ver2]
```

In the log file, Spectre FX displays the updates related to the reading and backannotation of each pair of SPF/DPF files with the elapsed time, and also displays the combined statistical data related to the backannotation of all the files, as shown below.

```
Finished reading and back-annotating test.spf, test.dpf (00:00:00.011)
Finished reading and back-annotating test_ver2.spf, test_ver2.dpf (00:00:00.011)
DSPF Parsing Summary: (00:00:00.022)
                Bytes parsed: 15633
                Number nets: 10
                Number of elements: 264
                       Number of MOSFETs: 24
                       Number of resistors: 120
                       Number of capacitors: 120
                               Number of grounded capacitors: 82
                               Number of coupled capacitors: 38
                Number of shorted resistors: 8

Backannotation Summary: (00:00:00.000)
                Number of nets: 10 (100.0%)
                Number of elements: 264 (100.0%)
                        Number of MOSFETS: 24 (100.0%)
                        Number of resistors: 120 (100.0%)
                        Number of capacitors: 120 (100.0%)
                               Number of grounded capacitors: 82
                               Number of coupling capacitors: 38
```

Example 2: A 20GB DSPF File

In the netlist, no subcircuit or instance is specified for the DSPF file. As a result, the subcircuit definition in the DSPF file is used. To read both RC NET and Instance sections, use the following two options for the same DSPF file:

```
.option sfx_ba_dspf_net=./netlist/all.dspf
.option sfx_ba_dspf_instance=./netlist/all.dspf
```

In the log file, while parsing every 1GB file or backannotating every 10M elements, Spectre FX updates the following information to indicate the status:

```
Starting Hierarchical Netlist Creation ...
Starting DSPF parsing (./netlist/all.dspf)...
        1 GB parsed...
        2 GB parsed..
        3 GB parsed...
        ……
        19 GB parsed...
Starting backannotation...
        10000000 elements backannotated...
        20000000 elements backnnotated...
        30000000 elements backannotated...
        ……
        100000000 elements backannotated...


Finished reading and backannotating ./netlist/all.dspf (00:19:10.592)
DSPF Parsing Summary: (00:10:36.143)
         Bytes parsed: 21130434084
         Number of nets: 1304157
         Number of elements: 108419360
               Number of diodes: 426
         Number of instances: 3532336
         Number of resistors: 17484138
         Number of capacitors: 87402460
                     Number of grounded capacitors: 12017964
                     Number of coupled capacitors: 75384496
         Number of shorted resistors: 47

Backannotation Summary: (00:08:34.448)

        Number of nets: 1153843 (88.5%)
         Number of floating nets: 114763 (8.8%)
         Number of elements: 104189868 (96.1%)
         Number of instances: 3532336 (100.0%)
                Number of resistors: 16787246 (96.0%)
                Number of capacitors: 83870286 (96.0%)
                        Number of grounded capacitors: 11594495
                        Number of coupling capacitors: 72275791

Time for Hierarchical Netlist Creation: CPU = 1.79657 ks (29m  56.6s), elapsed =
1.20154 ks (20m  1.5s).
```

Example 3: A Small DSPF File with Discarded Nets

The following example shows how the discarded nets are reported in the DSPF backannotation summary:

```
Finished reading and back-annotating test_discard.spf, test.dpf (00:00:00.007)
DSPF Parsing Summary: (00:00:00.007)
               Bytes parsed: 7767
               Number nets: 4
               Number discarded nets: 1
               Number elements: 103
                        Number MOSFETs: 12
                        Number resistors: 49
```

```
                        Number capacitors: 42
                                Number grounded capacitors: 28
                                 Number coupled capacitors: 14

Backannotation Summary: (00:00:00.000)
               Number nets: 4 (100.0%)
               Number elements: 96 (93.2%)
                          Number MOSFETS: 12 (100.0%)
                          Number resistors: 49 (100.0%)
                          Number capacitors: 35 (83.3%)
                                Number grounded capacitors: 28
                                Number coupling capacitors: 7
```

Related Topics

Post-layout Simulation Methodologies

The Parasitic Backannotation Flow

Control Options for the Parasitic Backannotation Flow

Guidelines for Parasitic Backannotation Options

8

Circuit Checks

Circuit checks enable you to analyze typical design problems, such as high impedance nodes, leakage paths between power supplies, timing errors, power issues, connectivity problems, or extreme rise and fall times. They can be separated into dynamic and static checks. Dynamic checks are performed during transient analysis. Static checks are topology checks that do not require any simulation.

Related Topics

Circuit Check Scoping

Output Format of Circuit Checks

Circuit Check Syntax

Dynamic Checks

Static Checks

Circuit Check Scoping

Most circuit checks can be applied either globally to the entire design, or locally to specific blocks of a design. The scoping options are available to define the scope of each circuit check. The circuit checks can be applied to a specific subcircuit instance (`inst`), or all instances of a subcircuit definition (`subckt`). Exclusion of a specific subcircuit instance (`xinst`), or all instances of a subcircuit (`xsubckt`) is supported. All scoping options support wildcards, and the hierarchy level of the scope can be defined by the `depth` option.

`node`	Non-hierarchical node name. The default value is none.
`net`	Hierarchical node names between which the leakage path is checked. Wildcards are not supported. The default value is none.
`dev`	Non-hierarchical element name. The default value is none.
`model`	Non-hierarchical model name. The default value is none.
`inst`	Subcircuit instances to which the circuit check is applied. The default value is `inst=*`.
`xinst`	Subcircuit instances that are excluded from the circuit check. The default value is none.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. The default value is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit that are excluded from the circuit checks. The default value is none.
`depth`	Hierarchy level (from the top level) to be checked. For example, `depth=3` reports errors from top level plus 3 sublevels. The default value is 8.

The `inst` and `xinst` arguments can be used only to specify subcircuit instances and not device instances. Combination of `inst`/`xinst` and `subckt`/`xsubckt` is not allowed.

The following wildcards are allowed for `node`, `dev`, `model`, `inst`, `xinst`, `subckt`, and `xsubckt`:

`*` – matches any character including an empty string and . (dot)

`?` – matches any single character including space and . (dot)

Related Topics

Output Format of Circuit Checks

Output Format of Circuit Checks

The default output format of all circuit checks is XML. Therefore, the output files have an extension `.xml` and are present in the `raw` directory.

You can use the `check_format` parameter with the `options` statement to convert the XML files to a text format, as shown below.

```
opt options check_format=text (Spectre format)
.option check_format=text (SPICE format)
```

After the conversion, the converted files in the text format have an extension `.rpt` and are present in the `raw` directory. The prefix of these files is the same as the prefix of the XML files.

The `check_format` parameter can be assigned one of the following possible values:

- `xml`: Saves the output in the xml and sqldb formats
- `text`: Saves the output in the text and sqldb formats
- `csv`: Saves the output in the csv and sqldb formats
- `sql`: Saves the output in the sqldb format
- `all`: Saves the output in all above-mentioned formats

Related Topics

Circuit Check Syntax

Circuit Check Syntax

All circuit checks follow the Spectre syntax and can be written either in Spectre syntax or SPICE syntax by adding `.cck` before the circuit check statement. If you are using the design checks in a SPICE format netlist, you must specify `simulator lang=spectre` before any circuit check statement.

For example, the following is the syntax for the `dyn_highz` check in Spectre format with line continuation:

```
simulator lang=spectre
chk1 dyn_highz node=[*] inst=[alpha beta \
gamma pll]
```

The following is the syntax in SPICE format:

```
simulator lang=spice
.cck chk1 dyn_highz node=[*] inst=[alpha beta gamma pll]
```

Note: The example results for the circuit checks are shown in Spectre syntax. In Spectre FX, you just need to add `.cck` at the beginning of the check statement to convert it to SPICE syntax.

Related Topics

Dynamic Checks

Static Checks

Dynamic Checks

Dynamic checks are performed during the transient simulation and are stimuli dependent. The checks use the `dyn_` prefix as part of the circuit check keyword and write the report to a file with the extension `.dynamic.xml`. The XML file can be viewed with any Web browser.

The following dynamic checks are supported in Spectre FX:

- Dynamic Active Node Check (dyn_actnode)
- Dynamic Capacitor Voltage Check (dyn_capv)
- Dynamic DC Leakage Current Path Check (dyn_dcpath)
- Dynamic Delay Check (dyn_delay)
- Dynamic Diode Voltage Check (dyn_diodev)
- Dynamic Excessive Element Current Check (dyn_exi)
- Dynamic Excessive Rise and Fall Time Check (dyn_exrf)
- Dynamic Glitch Check (dyn_glitch)
- Dynamic Float Crosstalk Check (dyn_floatxtalk)
- Dynamic Floating Gate Induced Leakage Check (dyn_floatdcpath)
- Dynamic High Impedance Node Check (dyn_highz)
- Dynamic MOSFET Voltage Check (dyn_mosv)
- Dynamic Node Capacitance Check (dyn_nodecap)
- Dynamic Pulse Width Check (dyn_pulsewidth)
- Dynamic Setup and Hold Check (dyn_setuphold)
- Dynamic Subcircuit Port Power Check (dyn_subcktpwr)

Dynamic Active Node Check (dyn_actnode)

Spectre Syntax

```
title dyn_actnode node=[n1 n2 ...] dv=<value> type=<value> time_window=[start1
     stop1 start2 stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <fanout=all|gate|bulk> <depth=n>
     error_limit=<value>
```

SPICE Syntax

```
.cck title dyn_actnode node=[n1 n2 ...] dv=<value> type=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <fanout=all|gate|bulk> <depth=n> error_limit=<value>
```

Description

The dynamic active node check detects nodes with voltage changes that exceed the user-defined threshold `dv`. The voltage change is defined as peak-to-peak voltage (Vpp) within a time window.

The results are written to the `dynamic.xml` file, which can be viewed in a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`dv`	Voltage change threshold for the active nodes. Default is 0.1 volt.
`type`	Report inactive or active nodes or both. Possible values are `act`, `inact`, and `both`.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`error_limit`	Maximum number of errors to be reported. Default is 10000.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.

Example

Spectre Syntax

```
chk2 dyn_actnode node=[*] dv=4 type=act time_window=[0 3e-07]
```

SPICE Syntax

```
.cck chk2 dyn_actnode node=[*] dv=4 type=act time_window=[0 3e-07]
```

The above command reports all nodes that are active. These active nodes will have peak-to-peak voltage above 4V between 0s to 300ns. In addition, the command prints a summary

report that lists the percentage of active nodes having peak-to-peak voltage above 4V between 0s to 300ns.

The following is an example of the report that is displayed on the Web browser.

dyn_actnode: chk1

- chk1 dyn_actnode node=["*"] dv=4 type=act time_window=[0 3e-07]
- Violation Count: 7

Dynamic Active Node Check - Summary

Title	Percentage of active nodes	From(s)	To(s)
chk1	7.792%	0.000000e+00	3.000000e-07

Dynamic Active Node Check - Active Nodes

Title	Node Name	Vpp(V)	From(s)	To(s)
chk1	X5.N2N1484	4.125149e+00	0.000000e+00	3.000000e-07
chk1	X5.N2N1485	5.344194e+00	0.000000e+00	3.000000e-07
chk1	X5.N2N1486	5.016927e+00	0.000000e+00	3.000000e-07
chk1	X5.N2N1489	4.030851e+00	0.000000e+00	3.000000e-07
chk1	X5.NA	4.235771e+00	0.000000e+00	3.000000e-07
chk1	X5.ND	4.161615e+00	0.000000e+00	3.000000e-07

Related Topics

Dynamic Checks

Dynamic Capacitor Voltage Check (dyn_capv)

Spectre Syntax

```
title dyn_capv cond=<expression> duration=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
```

SPICE Syntax

```
.cck title dyn_capv cond=<expression> duration=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
```

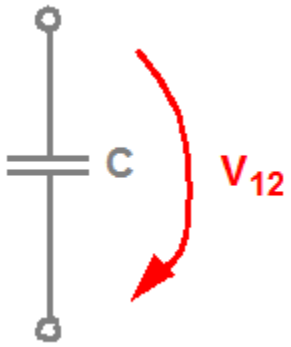
Description

Reports capacitor elements fulfilling the conditional expression on element voltages for a duration longer than the user-defined duration threshold (`duration`).

Supported capacitor variables are: `V(1,2)`, `V(1)`, and `V(2)`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.



Arguments

`cond`	Condition to be checked (default is `none`).

`duration`	Conditions with duration longer than the specified value are reported (default is `5ns`).
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
capv dyn_capv cond="V(1, 2)>0.2” duration=0.5n
```

SPICE Syntax

```
.cck capv dyn_capv cond="V(1, 2)>0.2" duration=0.5n
```

The above command will report the capacitor elements that fulfill the condition `V(1,2)>0.2V` for a duration longer than `5e-10s`. The following is an example of the report that is displayed in the Web browser:

Dynamic Capacitor Voltage Check Violations

dyn_capv: capv2

capv2 dyn_capv cond="v(1,2)>0.2" duration=5e-10

Violation Count: 1

Title	Instance Name	Start(s)	Model Name	Term1 Name	Term2 Name	Duration(s)	Dissatisfied	Violating Value	Check Time(s)
capv2	c1	1.011000e-09	capacitor	in	in1	6.850000e-10	(v(1, 2) > (0.2))	v(1, 2)=0.206731	1.011000e-09

Related Topics

Dynamic Checks

Dynamic DC Leakage Current Path Check (dyn_dcpath)

Spectre Syntax

```
title dyn_dcpath net=[n1 n2 ...] duration=<value> ith=<value> time_window=[start1
     stop1 start2 stop2 ....] <save=no|violation> <spice=no|yes>
     <leaki_times=[t1 t2 …]> <xinst=[xinst1 xinst2...]> error_limit=<value>
     <sort=no|current>
```

SPICE Syntax

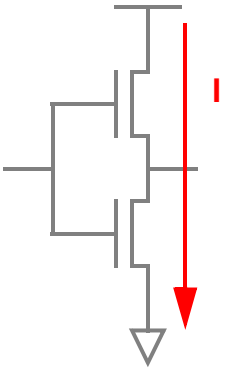
```
.cck title dyn_dcpath net=[n1 n2 ...] duration=<value> ith=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes> <leaki_times=[t1 t2 …]> <xinst=[xinst1 xinst2...]>
     error_limit=<value> <sort=no|current>
```

Description

Reports the conductance paths between user-specified nets. The check is based on transient analysis and reports th qualifying paths carrying an absolute current higher than the parameter `ith` for a duration longer than the specified `duration`, within the specified `time_window`. If you specify `leaki_times`, Spectre FX treats it like a special time window with only one time point.

If more than two nets are specified, Spectre FX checks the leakage path between each net combination. For example, if `net=[vdc1 vdc2 0]` is specified, then the conducting path between `vdc1` and `vdc2`, `vdc1` and `0`, and `vdc2` and `0` is checked.

The results are written to a file with the extension `dynamic.xml`.



Arguments

`net`	Hierarchical node names between which the leakage path is checked. Wildcarding is not allowed. All combinations of nets are checked. Default is none.
`duration`	Leakage paths with a duration longer than specified value are reported (default is `5ns`).
`sort`	If set to `no`, sorting is not performed. If set to `current`, the violations are sorted by max current. Possible values are `no` and `current`. The default value is `no`.
`ith`	Leakage paths with absolute current higher than the specified value are reported (default is `50uA`).
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`leaki_times`	Defines the times at which the check is performed.
`xinst`	Subcircuit instances to be excluded from the check. Default is none.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
dcpath1 dyn_dcpath ith=200u net=[vdd 0] duration=1n time_window=[1n 20n]
```

SPICE Syntax

```
.cck dcpath1 dyn_dcpath ith=200u net=[vdd 0] duration=1n time_window=[1n 20n]
```

The above command will report the DC conductance path between `vdd` and `0`. The command will report the paths that carry current higher than `200u` for a duration longer than `1n` within the time window between `1n` and `20n`. The following is an example of the report that is displayed in the Web browser:

Dynamic DC Leakage Path Check Violations

dyn_dcpath: dcpath1

dcpath1 dyn_dcpath ith=0.0002 duration=1e-09 net=["vdd" "0"] time_window=[1e-09 2e-08]

Violation Count: 1

Dynamic dcpath check based on transient analysis - Violation

Title	Path Id	From Net	To Net	Start(s)	Duration(s)
dcpath1	0	vdd	0	1.033403e-08	4.944546e-09

Dcpath datas: size=2

Instance	Current(A)
MP1	2.944546e-04
MN1	2.363355e-04

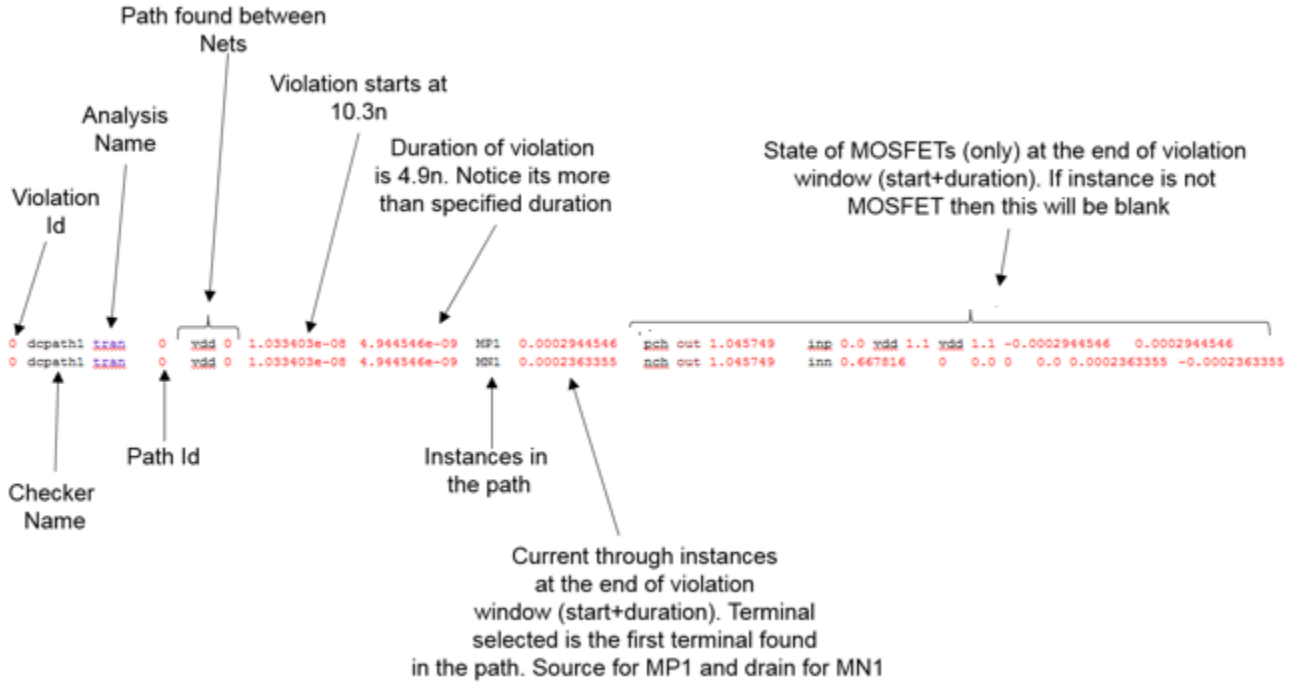
Elemstate datas: size=2

Instance Name	Model	Drain Name	Drain Volt(V)	Gate Name	Gate Volt(V)	Source Name	Source Volt (V)	Bulk Name	Bulk Volt(V)	Drain Current (A)	Source Current(A)
MP1	pch	out	1.045749e+00	inp	0.000000e+00	vdd	1.100000e+00	vdd	1.100000e+00	-2.944546e-04	2.944546e-04
MN1	nch	out	1.045749e+00	inn	6.678160e-01	0	0.000000e+00	0	0.000000e+00	2.363355e-04	-2.363355e-04

The path report has three sections. Each section is explained below.

- The first section shows that a path is found starting from net `vdd` to net `0`. Since this is a check based on transient-analysis, the name of the table is *Dynamic dcpath check based on transient analysis*.
- The second section shows the actual instances in the path. The first column *Instance* contains instances in the path. The second column contains the current through the instances. The current is taken at the end of violation, that is, at 1.52e-8 seconds (start + duration = 1.03e-8 + 4.94e-9).
- The third section shows the state of MOSFETs in the path. It does not report anything other than MOSFETs. The first column *Instance Name* is the MOSFET name. The second column is the model name. The following columns are the terminal names and voltages. The last two columns show the drain and source terminal currents. The values are taken at the end of violation.

Similar to the XML report, the text report is shown below:



Related Topics

Dynamic Checks

Dynamic Delay Check (dyn_delay)

Spectre Syntax

```
title dyn_delay node=[node] ref_node=[node] min_time=<value> max_time=<value>
     edge=[rise|fall|both] ref_edge=[rise|fall|both] vlth=<value>
     ref_vlth=<value> vhth=<value> ref_vhth=<value> time_window=[ start1 stop1
     start2 stop2 .... ] <save=no|violation> <spice=no|yes> <subckt=[subckt1]>
     <fanout=all|gate|bulk> error_limit=<value>
```

SPICE Syntax

```
.cck title dyn_delay node=[node] ref_node=[node] min_time=<value>
     max_time=<value> edge=[rise|fall|both] ref_edge=[rise|fall|both]
     vlth=<value> ref_vlth=<value> vhth=<value> ref_vhth=<value> time_window=[
     start1 stop1 start2 stop2 .... ] <save=no|violation> <spice=no|yes>
     <subckt=[subckt1]> <fanout=all|gate|bulk> error_limit=<value>
```

Description

Checks timing delays between two signals and reports nodes with edge delay errors.

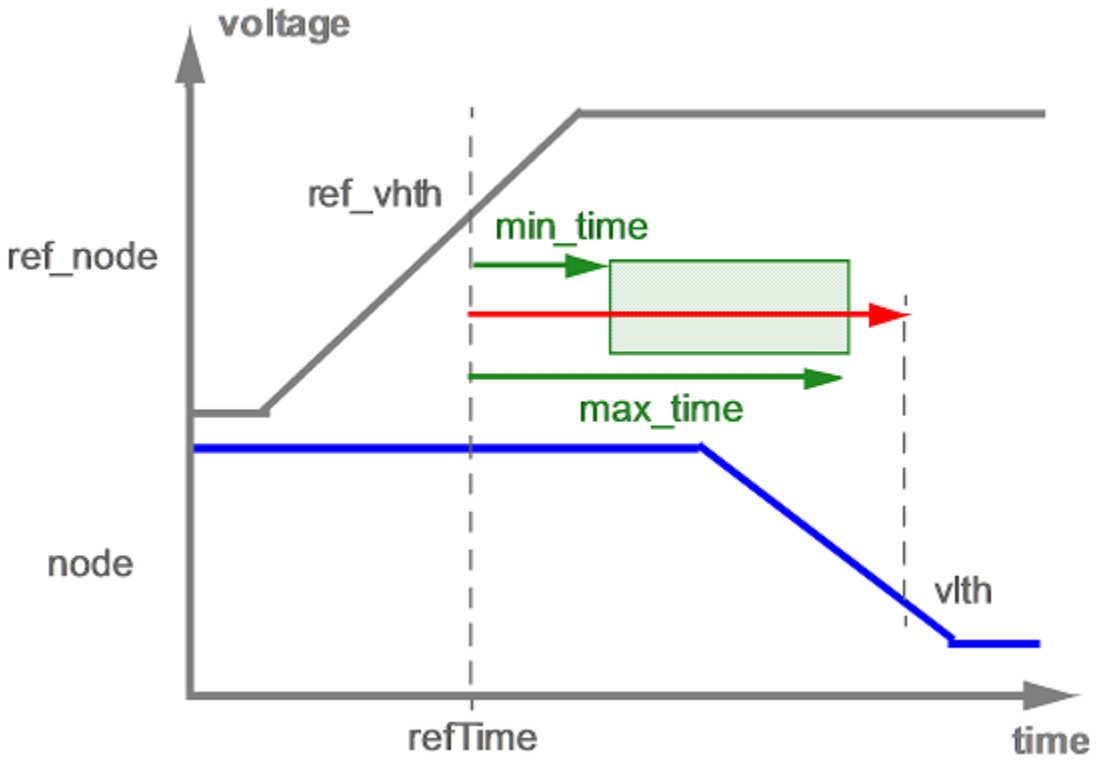
The delay is measured between user-specified nodes and a reference node. A timing delay error occurs when the transition time of a signal falls outside the range of `refTime + min_time` and `refTime + max_time`, where `refTime` is the transition time of the reference signal. In other words, a timing delay error occurs when the delay between the signal and the reference signal is outside the range of `min_time` and `max_time`.

The `ref_vhth` and `vhth` parameters are used for triggering rising edge measurements, while `ref_vlth` and `vlth` parameters are used for triggering falling edge measurements. For example, a delay measurement from a rising reference signal to a falling signal includes measuring the delay from the time the reference signal crosses `ref_vhth` to the time the signal crosses `vlth`.

If the `subckt` parameter is specified, then `node` and `ref_node` are considered as local nodes to the specified subcircuit. In other words, `node` and `ref_node` belong to the instances of the specified subcircuit. Only one `subckt` value can be specified per check, with no wildcard.

If the `subckt` parameter is not specified, `node` and `ref_node` are considered as global nodes with hierarchical names starting from the top level.

The results are written to the `dynamic.xml` file, which can be viewed in a Web browser.



In the figure above, an error is reported if the signal net transition occurs outside the green marked area.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`ref_node`	Name of the referenced node. Wildcards are not supported.
`min_time`	Minimum time delay between the signal and the reference signal transition. The specified value can be negative, 0, or positive.
`max_time`	Maximum time delay between the signal and the reference signal transition. The specified value can be negative, 0, or positive.
`edge`	Edge type of the signal net. Possible values are `rise`, `fall`, and `both`. Default value is `rise`.
`ref_edge`	Edge type of the reference signal. Possible values are `rise`, `fall`, and `both`. Default value is `rise`.
`vlth`	Low voltage threshold for the signal net.

`ref_vlth`	Low voltage threshold for the referenced net.
`vhth`	High voltage threshold for the signal net.
`ref_vhth`	High voltage threshold for the referenced net.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`error_limit`	Maximum number of errors reported. Default is 10000.
`subckt`	Instances of the specified subcircuit to which the check is applied. Default is none.

Example

Spectre Syntax

```
d1 dyn_delay ref_node=out8 node=out1 ref_edge=rise edge=fall min_time=100p
max_time=5n time_window=[115n 200n]
```

SPICE Syntax

```
.cck d1 dyn_delay ref_node=out8 node=out1 ref_edge=rise edge=fall min_time=100p
max_time=5n time_window=[115n 200n]
```

The above command reports any transition of signal data having delay of more than `5n` seconds or having delay of less than `100p` seconds. The delay is evaluated from the rising edge of `out8` node to the falling edge of `out1` node.

If the `subckt` parameter is not specified, then `node` and `ref_node` are considered as global nodes with the hierarchical names starting from the top level.

The following is an example of the report that is displayed on the Web browser:

Dynamic Delay Check Violations

dyn_delay: d1

- d1 dyn_delay ref_node="out8" node=["out1"] ref_edge=rise edge=fall min_time=1e-10 max_time=5e-09 time_window=[1.15e-07 2e-07]
- Violation Count: 1

Title	Ref Node Name	Node Name	Ref Edge	Edge	Reference Time(s)	Time(s)	Delay(s)
d1	out8	out1	rise	fall	1.195470e-07	1.911900e-07	7.164300e-08

Related Topics

Dynamic Checks

Dynamic Diode Voltage Check (dyn_diodev)

Spectre Syntax

```
title dyn_diodev model=[m1 m2 ...] cond=<expression> duration=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> <depth=n>
     error_limit=<value>
```

SPICE Syntax

```
.cck title dyn_diodev model=[m1 m2 ...] cond=<expression> duration=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> <depth=n>
     error_limit=<value>
```

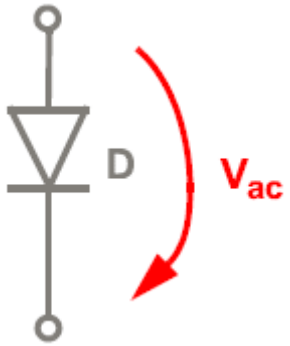
Description

Reports diode devices fulfilling the conditional expression on device voltages for a duration longer than the user-defined duration threshold (`duration`).

Supported diode variables are: `v(a,c)`, `v(a)`, and `v(c)`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.



Arguments

`model`	Diode device model names to be checked.
`cond`	Condition to be checked (default is `none`).
`duration`	Conditions with duration longer than the specified value are reported (default is `5ns`).
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. depth=3 reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
dv1 dyn_diodev model=["diode1"] cond="v(a,c)>0.5" duration=2.5e-08
```

SPICE Syntax

```
.cck dv1 dyn_diodev model=["diode1"] cond="v(a,c)>0.5" duration=2.5e-08
```

The above command will report the instances of `diode1` that fulfill the condition `v(a,c)>0.5` for a duration longer than `25n`. The following is an example of the report that is displayed in the Web browser:

Dynamic Diode Voltage Check Violations

dyn_diodev: dv1

dv1 dyn_diodev model=["diode1"] cond="v(a,c)>0.5" duration=2.5e-08

Violation Count: 2

Title	Instance Name	Start(s)	Model Name	Anode Name	Cathode Name	Duration(s)	Dissatisfied	Violating Value	Check Time(s)
dv1	d2	3.968000e-09	diode1	mid	vdd	4.638400e-08	(v(a, c) > (0.5))	v(a, c)=0.535796	3.968000e-09
dv1	d1	5.169600e-08	diode1	0	mid	5.288000e-08	(v(a, c) > (0.5))	v(a, c)=0.510379	5.169600e-08

Related Topics

Dynamic Checks

Dynamic Excessive Element Current Check (dyn_exi)

Spectre Syntax

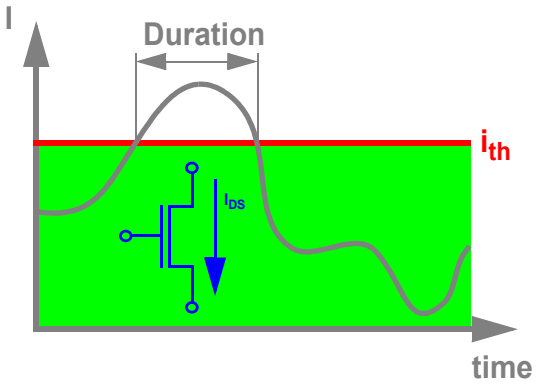
```
title dyn_exi dev=[d1 d2 ...] duration=<value> ith=<value> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes>
```

SPICE Syntax

```
.cck title dyn_exi dev=[d1 d2 ...] duration=<value> ith=<value> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes>
```

Description

Reports elements and devices carrying currents (absolute value) above a threshold specified by `ith` for a time longer than the duration threshold specified by `duration`. The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.



Arguments

`dev`	Device (MOSFET, R, C, and so on) instance names to be checked (default is `none`).
`duration`	Excessive currents with a duration longer than value are reported (default is `5ns`).

`ith`	Elements with current higher than the specified value are reported (default is none).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.

Example

Spectre Syntax

```
exi dyn_exi dev=["*"] ith=1e-06 duration=1e-09
```

SPICE Syntax

```
.cck exi dyn_exi dev=["*"] ith=1e-06 duration=1e-09
```

The above command will report all device instances that carry excessive currents higher than `1e-06` for a duration longer than `1e-09s`. The following is an example of the report that is displayed in the Web browser:

Dynamic Excessive Element Current Check Violations

dyn_exi: exi1

- exi1 dyn_exi dev=["*"] ith=1e-06 duration=1e-09
- Violation Count: 2

Title	Instance Name	Start(s)	Duration(s)	Max Current(A)
exi1	x1.mn1	4.923000e-09	5.077000e-09	2.320818e-03
exi1	x2.r1	1.280000e-10	9.872000e-09	3.000000e-03

Related Topics

Dynamic Checks

Dynamic Excessive Rise and Fall Time Check (dyn_exrf)

Spectre Syntax

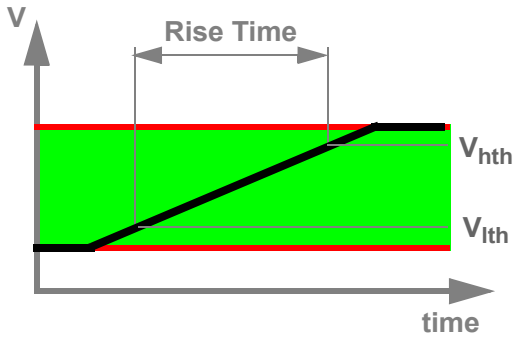
```
title dyn_exrf node=[n1 n2 ...] rise=<value> fall=<value> utime=<value>
     vlth=<value> vhth=<value> time_window=[start1 stop1 start2 stop2 ....]
     <save=no|violation> <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> <fanout=all|gate|bulk> <filter=[no|rc]> <inverse=[no|yes]>
     <sort=[no|duration]> <depth=n> error_limit=<value>
```

SPICE Syntax

```
.cck title dyn_exrf node=[n1 n2 ...] rise=<value> fall=<value> utime=<value>
     vlth=<value> vhth=<value> time_window=[start1 stop1 start2 stop2 ....]
     <save=no|violation> <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> <fanout=all|gate|bulk> <filter=[no|rc]> <inverse=[no|yes]>
     <sort=[no|duration]> <depth=n> error_limit=<value>
```

Description

Reports nodes with excessive rise and fall times, or nodes with an undefined state. The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.



Arguments

`node`	Nodes to be checked. Default is none.
`rise`	Rise times longer than specified value are reported (default is `none`). The rise time is measured between low (`vlth`) and high (`vhth`) voltage thresholds.

`fall`	Fall times longer than specified value are reported (default is `none`). The fall time is measured between low (`vlth`) and high (`vhth`) voltage thresholds.
`utime`	Undefined time threshold. Undefined state is defined by node voltages between the low and high voltage thresholds.
`vlth`	Low voltage threshold for `rise`, `fall`, and `utime` measurements (default is `none`).
`vhth`	High voltage threshold for `rise`, `fall`, and `utime` measurements (default is `none`).
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`.
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`.

`filter`	If set to `no`, filtering is not performed. If set to `rc`, `dyn_exrf` checks only one node in the same parasitic sub. Possible values are `no` and `rc`.
`inverse`	If set to `no`, reports rise times and fall times longer than the specified value. If set to `yes`, reports rise times and fall times smaller than the specified value. Possible values are `no` and `yes`. Default is `no`.
`sort`	Sort the violations. If set to `no`, no sorting is performed. If set to `duration`, violations are sorted based on duration. Possible values are `no` and `duration`. The default value is `no`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
exrf1 dyn_exrf node=["*"] rise=5e-10 fall=5e-10 utime=1e-09 vlth=0.3 vhth=2.7
time_window=[1e-09 9e-09]
```

SPICE Syntax

```
.cck exrf1 dyn_exrf node=["*"] rise=5e-10 fall=5e-10 utime=1e-09 vlth=0.3 vhth=2.7
time_window=[1e-09 9e-09]
```

The above command will report nodes with rise and fall times larger than `5e-10s` and nodes with undefined states longer than `1e-09s`. Measurements are performed between the voltage thresholds `0.3V` and `2.7V` and within the time window between `1e-09s` and `9e-09s`. The following is an example of the report that is displayed in the Web browser:

Dynamic Excessive Rise, Fall, Undefined State Time Check Violations

dyn_exrf: exrf1

- exrf1 dyn_exrf node=["*"] rise=5e-10 fall=5e-10 utime=1e-09 vlth=0.3 vhth=2.7 time_window=[1e-09 9e-09]
- Violation Count: 5

Title	Node Name	Type	Start(s)	Duration(s)
exrf1	out	fall	1.390000e-09	1.288000e-09
exrf1	out1	utime	1.440000e-09	7.560000e-09
exrf1	t1	rise	4.838000e-09	8.270000e-10
exrf1	t2	fall	5.596000e-09	8.870000e-10
exrf1	out	rise	6.374000e-09	1.336000e-09

Related Topics

Dynamic Checks

Dynamic Glitch Check (dyn_glitch)

Spectre Syntax

```
title dyn_glitch parameter=value ...
```

SPICE Syntax

```
.cck title dyn_glitch parameter=value ...
```

Description

A glitch occurs when:

- A low signal goes above the mid level, and crosses the mid level again in a time less than the user-defined duration.
- A high signal goes below mid level, and crosses the mid level again in a time less than the user-defined duration.

The check applies only to blocks with single and constant power supply.

The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`duration`	Duration threshold of a glitch (default is `5ns`).
`min_duration`	Minimum glitch duration threshold. Default is 0 sec.
`max_duration`	Maximum glitch duration threshold Default is `5ns`.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`high`	High voltage level (default is `none`).
`low`	Low voltage level (default is `0V`).
`mid`	Mid voltage level (default is `0.5*(high+low)`.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`. This option is supported only in Spectre FastSPICE mode.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Related Topics

Dynamic Checks

Dynamic Float Crosstalk Check (dyn_floatxtalk)

Spectre Syntax

```
title dyn_floatxtalk parameter=value ...
```

SPICE Syntax

```
.cck title dyn_floatxtalk parameter=value ...
```

Description

Reports the glitches of high impedance nodes caused by crosstalk of parasitic capacitors in DSPF file. A high impedance state occurs when there is no conducting path from the node to any power supply or ground.

The following device conducting rules are used for the floating node detection:

- MOSFET is conducting if mos region is triode or saturation
- Resistors, controlled resistors, phy_res, relay and inductors are conducting if R<=res_th
- BJT is conducting if Vbe>bjt_vbe OR Ic>bjt_ith
- Diode is conducting if V>diode_vth
- Vsources and iprobes are considered conducting
- Isources, VCCS, and CCCS are conducting if i>isource_ith
- JFET is considered conducting
- Mutual inductors and controlled capacitors are not conducting
- VerilogA: conducting path depends on module details

The results are written to the dynamic.xml file, which can be viewed in a Web browser.

Arguments

Design Check Parameters

`leaki_times`	Times at which the check is performed.
`rise_vth=0.2`	The threshold of voltage change caused by pulling up of an aggressor.
`fall_vth=0.2`	The threshold of voltage change caused by pulling down of an aggressor. Default is 0.2 V.
`cmin=1.0e-16`	If the lumped capacitance between highZ NET and its aggressor is less than cmin, Caggr<cmin, such aggressor is ignored. Default is 1.0e-16 F.
`capr=0.2`	If the ratio of between the lumped capacitance and the total capacitor of HighZ NET is less than capr, Caggr/Ctotal<capr, such aggressor is ignored. Default is 0.2.
`error_limit`	Maximum number of errors to be reported. Default is `10000`.

Floating Node Detection Parameters

`mos_ith`	Mosfet ids conducting threshold for high impedance node detection. Default is 100 nA.
`mos_gds`	Mosfet gds conducting threshold for floating node detection. Default is 1e-5.
`bjt_vbe`	BJT `vbe` conducting threshold for high impedance node detection. BJT is conducting if `i>bjt_ith` or `vbe>bjt_vbe`. Default is 0.4V.
`bjt_ith`	BJT `ith` conducting threshold for high impedance node detection. BJT is conducting if `i>bjt_ith` or `vbe>bjt_vbe`. Default is `50nA`.
`res_th_va`	Verilog-A resistor conducting threshold for high impedance node detection. This parameter is applicable only on AMS `ie` element, and on one or two-port AHDL module. Default is `10 MOhms`.
`res_th`	Resistor conducting threshold for high impedance node detection. Resistor is conducting if `R<res_th`. Default is `1TOhms`.
`diode_vth`	Diode voltage conducting threshold for high impedance node detection. Diode is conducting if `v>diode_vth`. Default is `0.6V`.

`isource_ith`	Current source conducting threshold for floating node detection. Current sources are conducting if `i>isource_ith`. Default is `1pA`.
`mos_conduct_rule=tri_sat`	The conducting rule to decide if a mosfet is on. Use tri_sat for triode or saturation region based rule. Use tri_sat_sub for triode or saturation or subthreshold region based rule. Use ids_gds for ids>mos_ith or gds>mos_gds rule. Use vgs for vgs>vth rule (Only supported in Fastspice). Possible values are tri_sat, tri_sat_sub, ids_gds, and vgs. Default is tri_sat. Possible values are `vgs`, `ids_gds`, `tri_sat` and `tri_sat_sub`.

Floating Node Detection Filtering Parameters

`node`	Nodes to which the floating node detection is applied. Default is `node=*`.
`xnode`	Nodes that are excluded from floating node detection. Default is `none`.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`.
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.

Related Topics

Dynamic Checks

Dynamic Floating Gate Induced Leakage Check (dyn_floatdcpath)

Spectre Syntax

```
title dyn_floatdcpath net=[n1 n2 ...] <leaki_times=[t1 t2 …]> ith=<value>
     error_limit=<value> <save=no|violation> <spice=no|yes>
     <detailed_path=[yes|per_fm|per_fn]> <sort=[no|current]> res_th_va=<value>
     res_th=<value> isource_ith=<value> bjt_vbe=<value> bjt_ith=<value>
     diode_vth=<value> <floatgate=[yes|no|gate_has_driver_no_cap]> node=[n1
     n2...] xnode=[n1 n2...] <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
```

SPICE Syntax

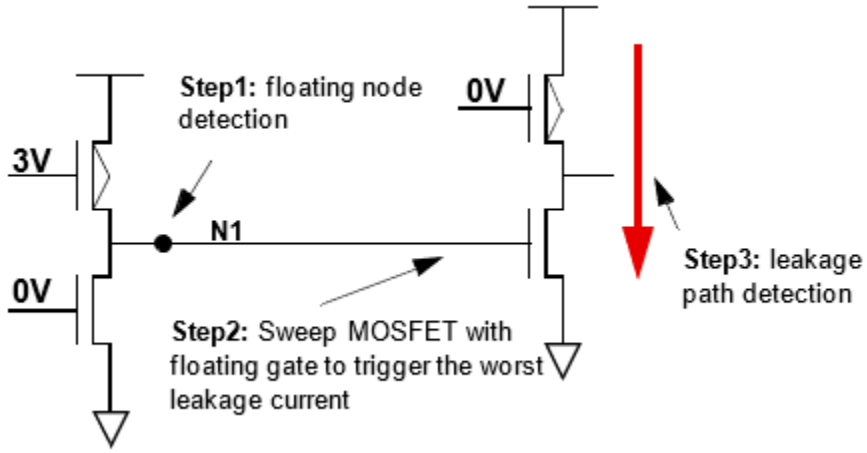
```
.cck title dyn_floatdcpath net=[n1 n2 ...] <leaki_times=[t1 t2 …]> ith=<value>
     error_limit=<value> <save=no|violation> <spice=no|yes>
     <detailed_path=[yes|per_fm|per_fn]> <sort=[no|current]> res_th_va=<value>
     res_th=<value> isource_ith=<value> bjt_vbe=<value> bjt_ith=<value>
     diode_vth=<value> <floatgate=[yes|no|gate_has_driver_no_cap]> node=[n1
     n2...] xnode=[n1 n2...] <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
```

Description

Reports the DC leakage paths that are caused by floating gate. The check based on leakage analysis is evoked when parameter `leaki_times` is specified. `leaki_times` should be carefully selected in standby or power-down mode. The check is performed for a transient simulation at the time points specified by `leaki_times`.

Worst leakage paths caused by floating gate MOSFET devices are estimated and reported. The leakage paths are checked between the specified power supply nodes (net).

The floating node detection and path detection in the leakage analysis-based method comprises of two steps.



Step 1: Floating node detection

This stage is like `dyn_higz` check. A node is considered floating (or in high impedance state) if it has no conducting path to a power supply or ground. A report of all MOSFETs with floating gates can be printed by using the parameter `floatgate`. The following device conducting rules are used for floating node detection:

- A MOSFET is considered to be conducting if `region` is either `triode` or `saturation`.
- Resistors, controlled resistors, phy_res, relay, and inductors are considered to be conducting if `R<= res_th`.
- A BJT is considered to be conducting if `vbe>bjt_vbe` or `ic>bjt_ith`.
- A diode is considered to be conducting if `v>diode_vth`.
- Vsource, and iprobes are considered to be conducting.
- Isources, VCCS, and CCCS are considered to be conducting if `i>isource_ith`.
- A JFET is considered to be conducting.
- Mutual inductors and controlled capacitors are considered to be non-conducting.
- In Verilog-A, conducting path depends on module details

Step 2: Current path detection

In the violation report of `dyn_floatdcpath`, the current is not from a transient simulation, but from the estimation of a worst scenario and the purpose is to trigger the violations.

The current path detection is like dyn_dcpath. If more than two nets are specified, Spectre checks the leakage path between each pair of nets. For example, if net=[vdc1 vdc2 0], the conducting path between vdc1 and vdc2, vdc1 and 0, and vdc2 and 0 is checked.

Arguments

General Parameters

`leaki_times` Times at which the check is performed.

`error_limit` Maximum number of errors to be reported. Default is `10000`.

`save` Specifies how to probe the signals specified within this circuit check. Possible values are:

- `no`: None of the signals in this circuit check are probed. This is the default value.
- `violation`: Automatically probe the signals with violations under this circuit check.

`spice` Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax.

Possible values are:

- `yes`: Saves the signals in SPICE syntax
- `no`: Saves the signals in Spectre syntax. This is the default value.

Path Detection Parameters

`net` Hierarchical node names between which the leakage path is checked. Wildcards are not allowed. All combination of nets are checked.

Default is `none`.

`ith` Leakage paths more than `ith` to be reported. The default value is `50uA`.

Path Detection Filtering Parameters

`detailed_path` Specifies how to print the path.

Possible values are:

- `yes`: Prints the detailed path.
- `per_fm`: Prints one path per floating MOSFET. This is the default value.
- `per_fn`: Print one path per floating node. This value is not supported in Spectre FastSPICE.

`sort` Specifies how to sort the violations.

Possible values are:

- `no`: No sorting is done. This is the default value.
- `current`: `dyn_floatdcpath` sorts the violations based on maximum current.

Floating Node Detection Parameters

`bjt_vbe` BJT `vbe` conducting threshold for high impedance node detection. BJT is conducting if `i>bjt_ith` or `vbe>bjt_vbe`.

Default is 0.4V.

`bjt_ith` BJT `ith` conducting threshold for high impedance node detection. BJT is conducting if `i>bjt_ith` or `vbe>bjt_vbe`. Default is `50nA`.

`res_th_va` Verilog-A resistor conducting threshold for high impedance node detection. This parameter is applicable only on AMS `ie` element, and on one or two-port AHDL module. Default is `10 MOhms`.

`res_th` Resistor conducting threshold for high impedance node detection. Resistor is conducting if `R<res_th`. Default is `1TOhms`.

`diode_vth` Diode voltage conducting threshold for high impedance node detection. Diode is conducting if `v>diode_vth`. Default is `0.6V`.

`isource_ith` Current source conducting threshold for floating node detection. Current sources are conducting if `i>isource_ith`. Default is `1pA`.

Floating Node Detection Filtering Parameters

`floatgate`	Specifies whether to report all MOSFETs with floating gate at specified `leaki_times`. Possible values are: ■ `gate_has_driver_no_moscap`: Floating MOSCAPs are not reported. ■ `yes`: All MOSFETs with floating gates at the specified `leaki_times` are reported. ■ `no`: The report is not generated. This is the default.
`node`	Nodes to which the floating node detection is applied. Default is `node=*`.
`xnode`	Nodes that are excluded from floating node detection. Default is `none`.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`.
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.

Example

```
dyn1 dyn_floatdcpath net=[vdd1 0] leaki_times=[7.5n]
```

This example reports the high impedance node induced DC leakage paths between the nets `vdd1` and `0` at the specified leaki times based on leakage analysis. This examples displays the following report in the Web browser.

dyn_floatdcpath: dyn1

- dyn1 dyn_floatdcpath net=["vdd1" "0"] leaki_times=[7.5e-09]
- Violation Count: 1

Dynamic Floating Node Induced DC Leakage Path Check - DC Path Violations

Title	From Net	To Net	Violation Time(s)
dyn1	vdd1	0	7.500000e-09

Fdcpath datas: size=3

Instance	Current(A)	Floating Gate Node	Floating Gate Node Volt(V)
MPA2	-2.886785e-06	N/A	0.000000e+00
MNA1	8.553140e-06	float	4.436366e-01
MNA2	8.551934e-06	N/A	0.000000e+00

Mosinfo datas: size=3

Device	Model	Drain Name	Drain Volt(V)	Gate Name	Gate Volt(V)	Source Name	Source Volt(V)	Bulk Name	Bulk Volt(V)
MPA2	bsim4	outA	1.098577e+00	0	0.000000e+00	vdd1	1.100000e+00	vdd1	1.100000e+00
MNA1	bsim4	outA	1.098577e+00	float	4.436366e-01	temp	2.054808e-03	0	0.000000e+00
MNA2	bsim4	temp	2.054808e-03	vdd1	1.100000e+00	0	0.000000e+00	0	0.000000e+00

The report contains the following three sections:

- The first section shows the From Net `vdd` To Net `0` between which the path is found.
- The second section shows the actual instances in the path. The first column `Instance` shows instances in the path. The second column `Current (A)` shows the current through the instances. The third column `Floating Gate Node` establishes the link between the floating gate node and MOSFET. The third column may contain either a node name or N/A. If it contains a node name, it means that the corresponding element is a MOSFET whose gate is floating. The fourth column shows the voltage of the floating gate.
- The third section shows the state of MOSFETs in the path. It does not report anything other than MOSFET. The first column `Device` is the MOSFET name. The second column `Model` shows the model name. The remaining columns show the terminal names and voltages.

Related Topics

Dynamic Checks

Dynamic High Impedance Node Check (dyn_highz)

Spectre Syntax

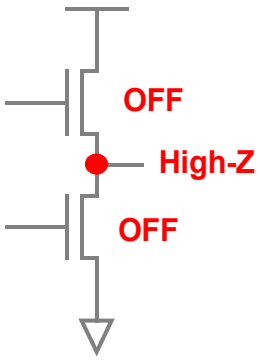
```
title dyn_highz  node=[n1 n2 ...] duration=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
     rest_th_va=<value> res_th=<value> isource_ith=<value> bjt_vbe=<value>
     bjt_ith=<value> diode_vth=<value> <inverse=no|yes> <sort=no|duration]
     <fanout=all|gate|bulk> <xnode=[node1 node2...]>
```

SPICE Syntax

```
.cck dyn_highz node=[n1 n2 ...] duration=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
     rest_th_va=<value> res_th=<value> isource_ith=<value> bjt_vbe=<value>
     bjt_ith=<value> diode_vth=<value> <inverse=no|yes> <sort=no|duration]
     <fanout=all|gate|bulk> <xnode=[node1 node2...]>
```

Description

Reports the nodes that are in high impedance state (also known as floating) for a duration longer than the user-defined threshold. A high impedance state occurs when there is no conducting path from the node to any power supply or ground.



The following device conditions are used:

- MOSFET is conducting if `region` is either `triode` or `saturation`.
- Resistors, controlled resistors, phy_res, relay, and inductors are conducting if `R<=res_th`.

- BJT is conducting if `vbe>bjt_vbe` or `ic>bjt_ith`.
- Diode is considered to be conducting if `v>diode_vth`.
- Vsource and iprobes are conducting.
- Isource, VCCS, and CCCS are conducting if `i>isource_ith`.
- JFET is considered to be conducting.
- Mutual inductor and controlled capacitors are considered to be non conducting.
- Verilog-A: Conducting path depends on the module details.

The results are written to the `dynamic.xml` file, which can be viewed in a Web browser.

Arguments

Design Check Parameters

`node`	Nodes to which the check is applied. Default is none.
`xnode`	Nodes to be excluded from the check. Default is none.
`duration`	HighZ states with a duration longer than value are reported (default is `5ns`).
`bjt_vbe`	BJT vbe conducting threshold. Default is 0.4V.
`bjt_ith`	BJT ith conducting threshold. Default is 50nA.
`res_th_va`	Verilog-A resistor conducting threshold for high impedance node detection. This parameter is applicable only on AMS `ie` element, and on one or two-port AHDL module. Default is 10 MOhms.
`res_th`	Resistor conducting threshold. Default is 1TOhms.
`isource_ith`	Current source conducting threshold. Default is 1pA.
`diode_vth`	Diode voltage conducting threshold. Default is 0.6V.
`inverse`	If set to `no`, reports all nodes that are in highz state. If set to `yes`, reports all nodes that are not in highz state. Possible values are `no` and `yes`. Default is `no`.
`sort`	Sort the violations. If set to `no`, no sorting is performed. If set to `duration`, violations are sorted based on `duration`. Possible values are `no` and `duration`. The default value is `no`.

Filtering Parameters

`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`error_limit`	Maximum number of errors to be reported. Default is 10000.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`.

Wildcard Scoping

`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.

Example

Spectre Syntax

```
hz1 dyn_highz node=["*"] duration=2e-09 time_window=[1e-09 1e-08]
```

SPICE Syntax

```
.cck hz1 dyn_highz node=["*"] duration=2e-09 time_window=[1e-09 1e-08]
```

The above command will report all nodes that were in a high impedance state for a duration longer than `2e-09s` within the time window between `1e-09s` and `1e-08s`. The following is an example of the report that is displayed in the Web browser:

Dynamic HighZ Node Check Violations

dyn_highz: hz1

- hz1 dyn_highz node=["*"] duration=2e-09 time_window=[1e-09 1e-08]
- Violation Count: 1

Title	Node Name	Start(s)	Duration(s)
hz1	out	1.000000e-09	9.000000e-09

Related Topics

Dynamic Checks

Dynamic MOSFET Voltage Check (dyn_mosv)

Spectre Syntax

```
title dyn_mosv model=[m1 m2 ...] cond=<expression> <sample=extreme|start>
     duration=<value> <sort=[no|duration]> time_window=[start1 stop1 start2
     stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
```

SPICE Syntax

```
.cck title dyn_mosv model=[m1 m2 ...] cond=<expression> <sample=extreme|start>
     duration=<value> <sort=[no|duration]> time_window=[start1 stop1 start2
     stop2 ....] <save=no|violation> <spice=no|yes> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
```

Description

Reports MOSFET devices fulfilling the conditional expression on device voltages and device size (`w`, `l`) for a duration longer than the user-defined duration threshold (`duration`).

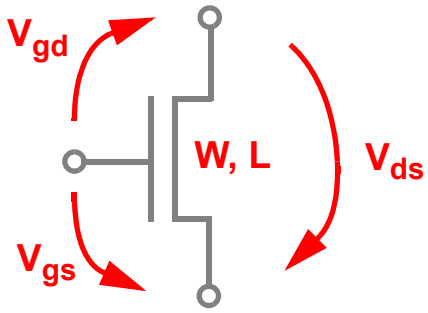
Supported MOSFET variables are: `v(g,s)`, `v(g,d)`, `v(g,b)`, `v(d,s)`, `v(d,b)`, `v(s,b)`, `v(g)`, `v(d)`, `v(s)`, `v(b)`, `l`, and `w`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

When `sample` is set to `extreme`, the extreme value of voltage function is reported. If the expression contains > or >=, the max value is reported as the extreme value. If the expression contains < or <= the min value is reported as the extreme value. Expressions having voltage function to the right and left of the comparison operator such as `v(g)>v(s)` are not supported. However, `v(g,s)>0` is supported.

In Spectre, Spectre APS, Spectre X, and Spectre FASTSPICE, the expression specified in `cond` can have multiple key expressions. For example, `cond="v(g,s)>0.4 && v(d,s)>0.4"`.

The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.



Arguments

`model`	MOSFET device model names to be checked.
`cond`	Condition to be checked (default is `none`).
`sample`	Report the extreme value or the start point. Possible values are `extreme` and `start`.
`sort`	Sort the violations. If set to no, no sorting is performed. If set to `duration`, violations are sorted based on duration. Possible values are `no` and `duration`. The default value is `no`.
`duration`	Conditions with duration longer than the specified value are reported (default is `5ns`).
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the voltage signals of four terminals and drain current, i(), of MOSFET for which violations are reported in this circuit check.

`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`.
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. depth=3 reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
mos1 dyn_mosv model=[nmos] cond=" v(g,s)>1.9" duration=2n
```

SPICE Syntax

```
.cck mos1 dyn_mosv model=[nmos] cond=" v(g,s)>1.9" duration=2n
```

The above command will report a MOSFET instance of NMOS that fulfills the condition `v(g,s)>1.9` for a duration longer than `2n`. The following is an example of the report that is displayed in the Web browser:

Dynamic MOSFET Voltage Check Violations

dyn_mosv: mos1

mos1 dyn_mosv model=["nmos"] cond="v(g,s)>1.9" duration=2e-09

Violation Count: 2

Dynamic mosv check - Violation

Title	Instance Name	Start(s)	Model Name	Drain Name	Gate Name	Source Name	Bulk Name	Duration(s)	Dissatisfied	Violating Value	Check Time(s
mos1	x1.mn2	1.058000e-09	nmos	mid	in	gnd	gnd	3.984000e-09	(v(g, s)>1.9)	v(g, s)=1.914000000000	1.058000e-09
mos1	x2.mn2	5.187000e-09	nmos	out	mid	gnd	gnd	4.813000e-09	(v(g, s)>1.9)	v(g, s)=1.909126953081	5.187000e-09

Dynamic Node Capacitance Check (dyn_nodecap)

Spectre Syntax

```
title dyn_nodecap <node=[node1 node2...]> <time=[t1 t2...]> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <fanout=all|gate|bulk> <depth=n>
     error_limit=<value><intrinsic_cap_merge=no|yes>
```

SPICE Syntax

```
.cck title dyn_nodecap <node=[node1 node2...]> <time=[t1 t2...]> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <fanout=all|gate|bulk> <depth=n>
     error_limit=<value><intrinsic_cap_merge=no|yes>
```

Description

Reports the node capacitance at the specified time (`time`) of a transient simulation. Device capacitances, voltage dependent capacitances, grounded and coupling caps are combined into one value. The results are written to a file with the extension `dynamic.xml`, which can be viewed using a Web browser.

Arguments

`node`	Nodes for which the capacitance needs to be checked. Default value is `none`.
`time`	Time points at which the check needs to be performed.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`.
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`intrinsic_cap_merge`	Merges the internal captab node with the external node. Possible values are `yes` and `no`. The default value is `no`. This option is supported only in Spectre and Spectre APS.

`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`. This option is supported only in Spectre FastSPICE mode.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports the errors on top level with three sublevels. Default is `8`.
`error_limit`	Maximum number of errors to be reported. Default is `10000`.

Example

```
n1 dyn_nodecap node=[*] time=[0 3n 7n]
```

This example reports the node capacitance for all nodes at times `0`, `3ns` and `7ns`.

The following is an example report displayed in the Web browser.

Dynamic Node Capacitance Check Violations

dyn_nodecap: n1

n1 dyn_nodecap node=["*"] time=[0 3e-09 7e-09]

Violation Count: 6

Title	Node Name	Time(s)	Capacitance(F)
n1	mid	0.000000e+00	3.239651e-13
n1	out	0.000000e+00	1.035941e-13
n1	mid	3.000000e-09	2.545755e-13
n1	out	3.000000e-09	1.982198e-13
n1	mid	7.000000e-09	3.234888e-13
n1	out	7.000000e-09	1.035941e-13

Related Topics

Dynamic Checks

Dynamic Pulse Width Check (dyn_pulsewidth)

Spectre Syntax

```
title dyn_pulsewidth node=[n1 n2 ...] pwmin_low=<value> pwmax_low=<value>
     pwmin_high=<value>  pwmax_high=<value> vlth=<value> vhth=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <fanout=all|gate|bulk> <depth=n> error_limit=<value>
```

SPICE Syntax

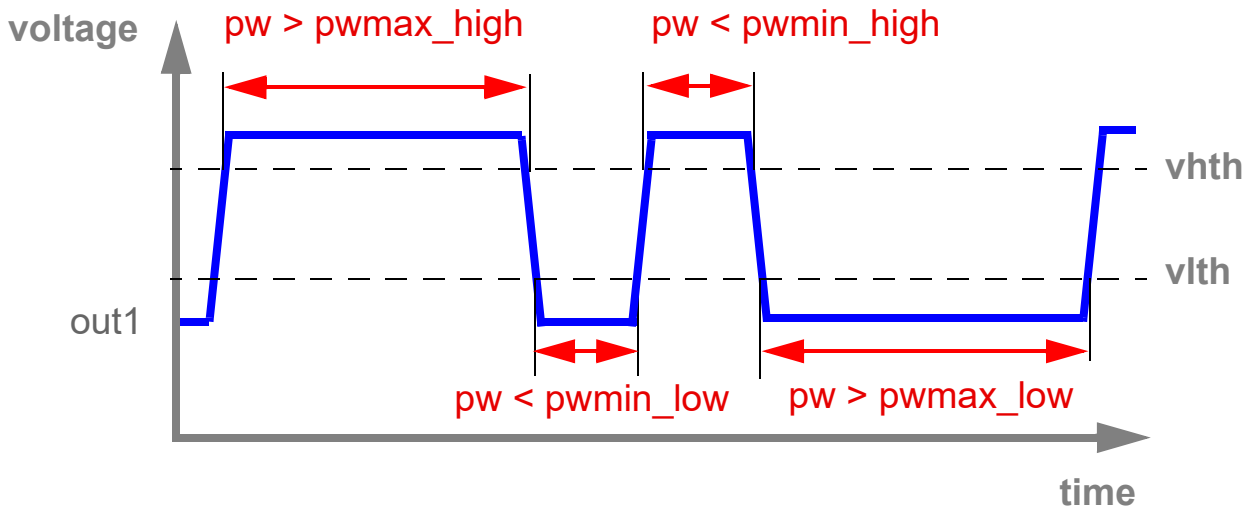
```
.cck title dyn_pulsewidth node=[n1 n2 ...] pwmin_low=<value> pwmax_low=<value>
     pwmin_high=<value>  pwmax_high=<value> vlth=<value> vhth=<value>
     time_window=[start1 stop1 start2 stop2 ....] <save=no|violation>
     <spice=no|yes> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <fanout=all|gate|bulk> <depth=n> error_limit=<value>
```

Description

The pulse width of a logic low state signal is the duration between which the signal crosses the low voltage threshold (`vlth`) while falling and again crosses `vlth` while rising. If this duration is outside the range specified using the `pwmin_low` and `pwmax_low` parameters, the `dyn_pulsewidth` check reports the pulse width of such signals.

Similarly, the pulse width of a logic high state signal is the duration between which the signal crosses the high voltage threshold (`vhth`) while rising and again crosses `vhth` while falling.

If this duration is outside the range of `pwmin_high` and `pwmax_high` parameters, the `dyn_pulsewidth` check reports the pulse width of such signals.



The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`pwmin_low`	Minimum value of the pulse width in logic low state. Default is 0.0 sec.
`pwmax_low`	Maximum value of the pulse width in logic low state. Default is infinity sec.
`pwmin_high`	Minimum value of the pulse width in logic high state. Default is 0.0 sec.
`pwmax_high`	Maximum value of the pulse width in logic high state. Default is infinity sec.
`vlth`	Low voltage threshold for the signal net. Default is 0.2v.
`vhth`	High voltage threshold for the signal net. Default is 0.8v.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`error_limit`	Maximum number of errors to be reported. Default is 10000.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.

Example

Spectre Syntax

```
chk1 dyn_pulsewidth node=[*] pwmin_low=20n pwmax_low=40n pwmin_high=20n
pwmax_high=40n vlth=0.2 vhth=1.0
```

SPICE Syntax

```
.cck chk1 dyn_pulsewidth node=[*] pwmin_low=20n pwmax_low=40n pwmin_high=20n
pwmax_high=40n vlth=0.2 vhth=1.0
```

The above command reports all nodes that specify either of the following conditions:

- The pulse width in logic low state is outside the range of parameters `pwmin_low` (20n) and `pwmax_low` (40n).
- The pulse width in logic high state is outside the range of parameters `pwmin_high` (20n) and `pwmax_high` (40n).

The following is an example of the report that is displayed in the Web browser.

Dynamic Pulse Width Check Violations

dyn_pulsewidth: chk1

chk1 dyn_pulsewidth node=["*"] pwmin_low=2e-08 pwmax_low=4e-08 pwmin_high=2e-08 pwmax_high=4e-08 vlth=0.2 vhth=1

Violation Count: 4

Title	Node Name	Type	Time(s)	Pulse Width(s)
chk1	out1	low	1.039700e-08	9.843000e-09
chk1	out1	high	2.035600e-08	4.994700e-08
chk1	out1	high	1.003570e-07	9.946000e-09
chk1	out1	low	1.103970e-07	4.984300e-08

Related Topics

Dynamic Checks

Dynamic Setup and Hold Check (dyn_setuphold)

Spectre Syntax

```
title dyn_setuphold node=[node] ref_node=[node] setup_time=<value>
     hold_time=<value> setup_chk_time=<value> setup_hold_time=<value>
     delay=<value> edge=[rise|fall|both] ref_edge=[rise|fall|both] vlth=<value>
     ref_vlth=<value> vhth=<value> ref_vhth=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> margin_stats=[yes|no]
     <subckt=[subckt1 subckt2....]> <fanout=all|gate|bulk> error_limit=<value>
     report=[violation|all]
```

SPICE Syntax

```
.cck title dyn_setuphold node=[node] ref_node=[node] setup_time=<value>
     hold_time=<value> setup_chk_time=<value> setup_hold_time=<value>
     delay=<value> edge=[rise|fall|both] ref_edge=[rise|fall|both] vlth=<value>
     ref_vlth=<value> vhth=<value> ref_vhth=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> margin_stats=[yes|no]
     <subckt=[subckt1 subckt2....]> <fanout=all|gate|bulk> error_limit=<value>
     report=[violation|all]
```

Description

Measures the timing of a signal net in comparison to a referenced (clock) net. It reports the setup or hold timing errors if the signal net transition happens within the specified violation window.

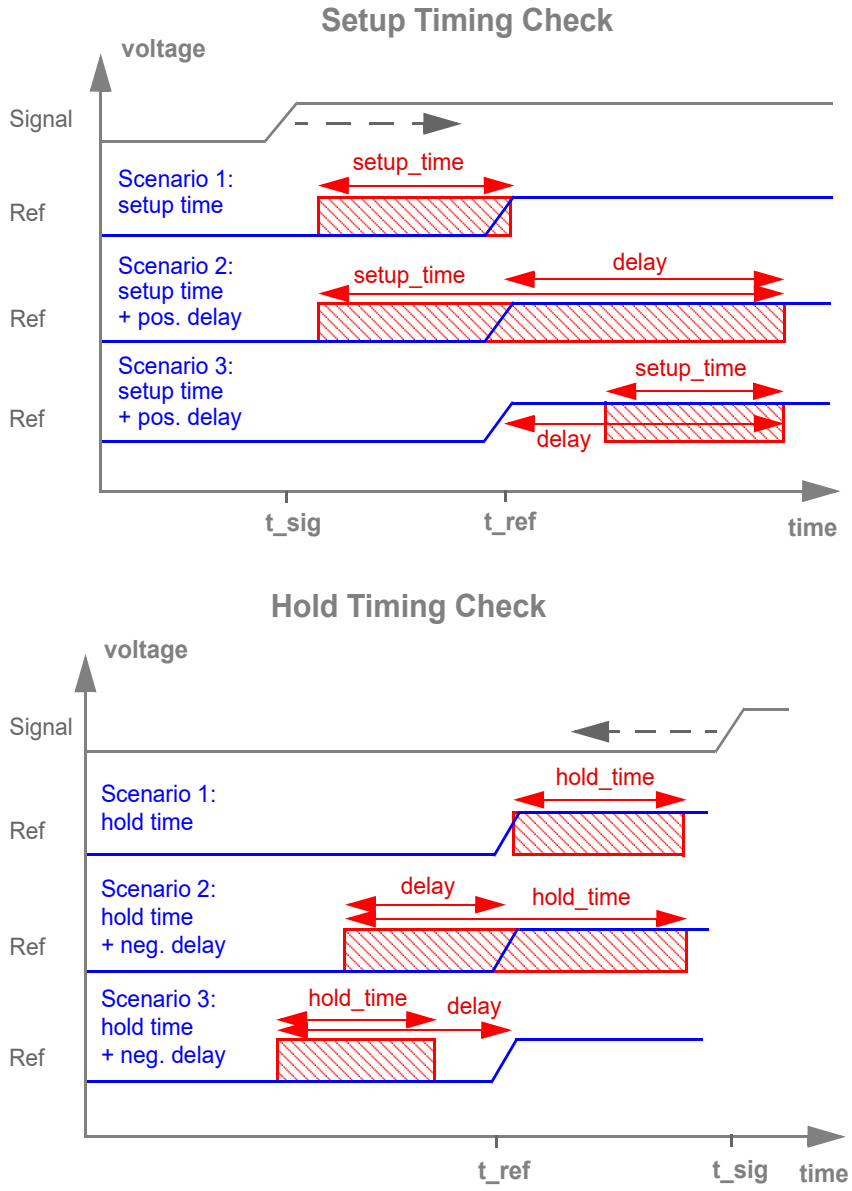
The violation window of the setup timing check is `refTime + delay - setup_time` and `refTime + delay`. The violation window for the hold timing check is `refTime + delay` and `refTime + delay + hold_time`. `refTime` is the transition time of the reference net.

`ref_vhth` and `vhth` parameters trigger the rising edge measurements, whereas `ref_vlth` and `vlth` parameters trigger the falling edge measurements.

If subcircuit parameter (`subckt`) is specified then the node (`node`) and reference node (`ref_node`) are considered local nodes to that subcircuit. That is, the nodes and reference nodes will belong to the instances of the specified subcircuit. Only one `subckt` value can be specified per check, with no wildcard.

If the `subckt` parameter is not specified then `node` and `ref_node` are considered as global nodes with hierarchical names starting from the top level.

The results are written to a file with the extension `dynamic.xml`, which can be viewed with a Web browser.



In the figure above, a setup or hold error is reported if the signal net transition occurs in the red marked area.

Arguments

`node`	Node to which the check is applied. Default is none.
`ref_node`	Name of the referenced clock (net). Wildcards are not supported.
`setup_time`	Setup time violation window. If specified, setup check is enabled. Default is 0.0 sec.
`hold_time`	Hold time violation window. If specified, hold check is enabled. Default is 0.0 sec.
`delay`	Delay time of the referenced signal. Default is 0.0 sec.
`edge`	Edge type of the signal net. Possible values are `rise`, `fall`, or `both`. Default is `rise`.
`ref_edge`	Edge type of the referenced signal net. Possible values are `rise`, `fall`, or `both`. Default is `rise`.
`setup_chk_time`	Specifies the setup time checking window. Spectre only checks the signal states within `refTime+delay-setup_chk_time` and `refTime+delay` window. Signal states outside the window are ignored. This argument is enabled only when setup check is enabled and the value of `report` is `all`. The specified value must be greater than `setup_time`. Default value is `2*setup_time`.
`hold_chk_time`	Specifies the hold time checking window. Spectre only checks the signal states within `refTime+delay+hold_chk_time` and `refTime+delay` window. Signal states outside the window are ignored. This argument is enabled only when hold check is enabled and the value of `report` is `all`. The specified value must be greater than `hold_time`. Default value is `2*hold_time`.
`vlth`	Low voltage threshold for the signal net. Default is 0.2v.
`ref_vlth`	Low voltage threshold for the referenced net. Default is 0.2v
`vhth`	High voltage threshold for the signal net. Default is 0.8v
`ref_vhth`	High voltage threshold for the referenced net. Default is 0.8v.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.

`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`margin_stats`	Specifies whether or not to print the margin statistics. By default, margin statistics are not printed. Possible values are `no` and `yes`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `none`.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`. This option is supported only in Spectre FastSPICE mode.
`report`	Reports all checks or violations only. Possible values are `all` and `violation`. Default value is `violation`.

Example 1

Spectre Syntax

```
s4 dyn_setuphold node=["*"] edge=rise ref_node="I9.I1.clk" ref_edge=rise
setup_time=5e-11 vhth=0.5 ref_vhth=0.5
```

SPICE Syntax

```
.cck s4 dyn_setuphold node=["*"] edge=rise ref_node="I9.I1.clk" ref_edge=rise
setup_time=5e-11 vhth=0.5 ref_vhth=0.5
```

The above command reports any transition of signal node in the time window between 0.5ns before the signal `clk` rises.

Since `subckt` parameter is not specified, it will compare all nodes with ref_node `I9.I1.sig_2` and report any violations.

The following is an example of the report that is displayed on the Web browser.

- s4 dyn_setuphold node=["*"] edge=rise ref_node="I9.I1.clk" ref_edge=rise setup_time=5e-11 vhth=0.5 ref_vhth=0.5
- Violation Count: 3

Title	Ref Node Name	Node Name	Type	Edge	Ref Edge	Time(s)	Reference Time(s)	Violation Window Start(s)	Violation Window End(s)
s4	I9.I1.clk	signal_2	setup	rise	rise	1.800000e-10	2.197383e-10	1.697383e-10	2.197383e-10
s4	I9.I1.clk	signal_4	setup	rise	rise	1.900000e-10	2.197383e-10	1.697383e-10	2.197383e-10
s4	I9.I1.clk	I9.I1.sig_1	setup	rise	rise	2.099673e-10	2.197383e-10	1.697383e-10	2.197383e-10

Example 2

Spectre Syntax

```
s1 dyn_setuphold node=["*"] edge=rise ref_node="clk" ref_edge=rise setup_time=5e-11 vhth=0.5 ref_vhth=0.5 subckt=ckt1
```

SPICE Syntax

```
.cck s1 dyn_setuphold node=["*"] edge=rise ref_node="clk" ref_edge=rise setup_time=5e-11 vhth=0.5 ref_vhth=0.5 subckt=ckt1
```

The above command reports any transition of the signal data in the time window between `0.5ns` before the signal `clk` rises.

Since `subckt=ckt1` is specified, it will compare all nodes in subckt `ckt1` with ref_node `sig_2` and report any violations. Note that `node` and `ref_node` belongs to same subckt `ckt1`.

The following is an example of the report that is displayed on the Web browser.

- s1 dyn_setuphold node=["*"] edge=rise ref_node="clk" ref_edge=rise setup_time=5e-11 vhth=0.5 ref_vhth=0.5 subckt=["ckt1"]
- Violation Count: 2

Title	Ref Node Name	Node Name	Type	Edge	Ref Edge	Time(s)	Reference Time(s)	Violation Window Start(s)	Violation Window End(s)
s1	I9.I1.clk	I9.I1.sig_1	setup	rise	rise	2.099673e-10	2.197383e-10	1.697383e-10	2.197383e-10
s1	I9.I0.clk	I9.I0.sig_1	setup	rise	rise	2.497238e-10	2.593195e-10	2.093195e-10	2.593195e-10

Related Topics

Dynamic Checks

Dynamic Subcircuit Port Power Check (dyn_subcktpwr)

Spectre Syntax

```
title dyn_subcktpwr <port=[port1 port2...]> <inst=[inst1 inst2...]>
     <power=[on|off]> error_limit=<value> time_window=[start1 stop1 start2
     stop2 ....] <save=no|violation> <spice=no|yes> <depth=n> filter=[none|gates]
     ith=<value>
```

SPICE Syntax

```
.cck title dyn_subcktpwr <port=[port1 port2...]> <inst=[inst1 inst2...]> <net=[n1
     n2...]> <power=[on|off]> error_limit=<value> time_window=[start1 stop1
     start2 stop2 ....] <save=no|violation> <spice=no|yes> <depth=n>
     filter=[none|gates] ith=<value>
```

Description

Reports port currents, port powers, and subcircuit powers.

The port current is positive when the current is going into a subcircuit. This check will report average, RMS, and maximum values of the current entering a port.

Any port name matching `inst` and `port` parameters are reported.

Power analysis can be done by using the parameter `power`. When the parameter `power` is set to `on`, then two additional sections are generated. The first section reports the average, RMS and the maximum power entering the ports, specified using the parameter `port`. The second section reports the average, RMS, and the maximum power consumed by each instance of a subcircuit, specified using the parameter `inst`.

Note: The wildcard in `port` parameter only considers the ports defined in the subcircuit definition. For global nodes, you need to add the ports manually.

Arguments

`port`	The ports that need to be analyzed. Default value is `none`.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`power`	If set to `off` (default), report only port currents. If set to `on`, report port currents, and power of ports and subcircuits.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is none.
`filter`	If set to `none` (default), all gates are checked. If set to `gates`, ports connected to MOSFET gates are skipped.
`error_limit`	Maximum number of errors to be reported. Default is 10000.
`time_window`	Time window to which the circuit check is applied. Multiple non-overlapping time windows are supported. Default time window is `0` to `tend`.
`save`	Specifies how to probe the signals specified within this circuit check. Possible values are: ■ `no`: None of the signals in this circuit check are probed. This is the default value. ■ `violation`: Automatically probe the signals with violations under this circuit check.
`spice`	Specifies whether to save the probed signals specified by the `save` parameter in the SPICE syntax. Possible values are: ■ `yes`: Saves the signals in SPICE syntax ■ `no`: Saves the signals in Spectre syntax. This is the default value.
`ith`	If all the `abs(AVG)`, `RMS` and `abs(MAX)` values are below `ith`, the values will be filtered out.

Example

Spectre Syntax

```
chk1 dyn_subcktpwr inst=[*] port=[*] depth=1 time_window=[0 10m] power=on
```

SPICE Syntax

```
.cck chk1 dyn_subcktpwr inst=[*] port=[*] depth=1 time_window=[0 10m] power=on
```

The above command reports the port current for all ports of all instances in the time window between 0 and 10ms. The report includes the current and power information for all subcircuit instances one level down the hierarchy.

The following is an example of the report that is displayed in the Web browser:

dyn_subcktpwr: chk1

- chk1 dyn_subcktpwr inst=["*"] port=["*"] depth=1 time_window=[0 0.01] power=on
- Violation Count: 5

Dynamic Subckt Port Power Check - Port Current Report

Title	Port Name	Avg(A)	RMS(A)	Max(A)	Max Time(s)	From(s)	To(s)
chk1	x_top.A	8.000015e-04	1.260086e-03	2.000004e-03	1.100000e-03	0.000000e+00	1.000000e-02
chk1	x_top.B	-8.000015e-04	1.260086e-03	-2.000004e-03	1.100000e-03	0.000000e+00	1.000000e-02

Dynamic Subckt Port Power Check - Port Power Report

Title	Port Name	Avg(W)	RMS(W)	Max(W)	Max Time(s)	From(s)	To(s)
chk1	x_top.A	7.939066e-04	1.255858e-03	2.000004e-03	1.100000e-03	0.000000e+00	1.000000e-02
chk1	x_top.B	7.939066e-04	1.255858e-03	2.000004e-03	1.100000e-03	0.000000e+00	1.000000e-02

Dynamic Subckt Port Power Check - Sum of Port Powers by Instance

Title	Instance Name	Avg(W)	RMS(W)	Max(W)	Max Time(s)	From(s)	To(s)
chk1	x_top	1.587813e-03	2.511717e-03	4.000008e-03	1.100000e-03	0.000000e+00	1.000000e-02

Related Topics

Dynamic Checks

Static Checks

Static checks are performed after parsing, using the topology information and voltage propagation. They apply to digital and SRAM circuits only and do not require a transient simulation. Static checks use the `static_` keyword prefix and write the results into a file with the extension `.static.xml`. The XML file can be viewed with any Web browser.

Note: Static checks are supported only in Spectre APS, Spectre X, Spectre XPS SPICE, Spectre XPS FastSPICE, Spectre FX and Spectre MS. These static circuit checks are not supported in Spectre.

The following static checks are supported by Spectre FX:

- Static Always Conducting NMOS/PMOS Check (static_nmosvgs/static_pmosvgs)
- Static Capacitor Check (static_capacitor)
- Static Capacitor Voltage Check (static_capv)
- Static Coupling Impact Check (static_coupling)
- Static DC Leakage Path Check (static_dcpath)
- Static Diode Voltage Check (static_diodev)
- Static ERC Check (static_erc)
- Static High Impedance Node Check (static_highz)
- Static Highfanout Check (static_highfanout)
- Static MOSFET Voltage Check (static_mosv)
- Static NMOS to VDD Count Check (static_nmos2vdd)
- Static NMOS/PMOS Forward Bias Bulk Check (static_nmosb/static_pmosb)
- Static PMOS to GND Count Check (static_pmos2gnd)
- Static RC Delay Check (static_rcdelay)
- Static Resistor Check (static_resistor)
- Static Resistor Voltage Check (static_resv)
- Static Subcircuit Port Voltage Check (static_subcktport)
- Static Transmission Gate Check (static_tgate)
- Static Voltage Domain Device Check (static_voltdomain)

Static Always Conducting NMOS/PMOS Check (static_nmosvgs/ static_pmosvgs)

Spectre Syntax

```
title static_nmosvgs model=[m1 m2 ...] vnth=<value> vpth=<value> vt=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>

title static_pmosvgs model=[m1 m2 ...] vnth=<value> vpth=<value> vt=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

SPICE Syntax

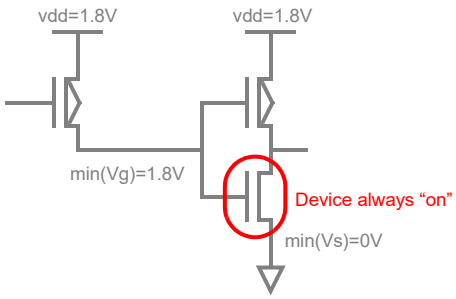
```
.cck title static_nmosvgs model=[m1 m2 ...] vnth=<value> vpth=<value> vt=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>

.cck title static_pmosvgs model=[m1 m2 ...] vnth=<value> vpth=<value> vt=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

Description

Reports MOSFET devices potentially always conducting due to connectivity problems. The following conditions are checked, and an error is reported if they are fulfilled:

- `NMOS:  min(Vg) > min (Vs/Vd) + abs(vt)`
- `PMOS:  max(Vg) < max (Vs/Vd) - abs(vt)`

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`model`	MOSFET device model names to be checked. Default is `none`.
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`vt`	MOSFET threshold voltage. Default is `0V`.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.

`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*`.nv`. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
mos1 static_nmosvgs model="nmos" vt=0.5
```

SPICE Syntax

```
.cck mos1 static_nmosvgs model="nmos" vt=0.5
```

The above command will check all instances of the nmos device for a potential "always on" state. The NMOS `vt` is defined with 500mV.

The following is an example of the report that is displayed in the Web browser:

Static Always Conducting MOSFET Check Violations

static_nmosvgs: mos1

mos1 static_nmosvgs model=["nmos"] vt=0.5

Violation Count: 1

Static Always Conducting MOSFET Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
mos1	x2.mn2	nmos	inv	(0, 0)	(1.8, 1.8)	(0, 0)	(0, 0)

Related Topics

Static Checks

Static Capacitor Check (static_capacitor)

Spectre Syntax

```
title static_capacitor type=[range|distr|print] cmin=<value> cmax=<value>
     error_limit=<value>
```

SPICE Syntax

```
.cck title static_capacitor type=[range|distr|print] cmin=<value> cmax=<value>
     error_limit=<value>
```

Description

Reports all capacitors within or outside the range `cmin` and `cmax`. Moreover, it also generates a distribution list of all capacitors in the circuit.

If the parameter type is set to `range`, then all capacitors outside the range of `cmin` and `cmax` will be reported.

If the parameter type is set to `print`, then all capacitors between `cmin` and `cmax` will be reported. The capacitor names can be sorted by clicking the *Device name* header.

If the parameter type is set to `distr`, then a distribution report will be generated for all capacitors. There are 9 bins that are:

```
-Inf - 0, 0 - 10a, 10a - 100a, 100a - 1f, 1f - 10f, 10f - 100f, 100f - 1p, 1p -
10p, 10p - Inf
```

However, if two or more consecutive bins are empty then they will merge into one bin, reducing the number of bins. The parameters `cmin`, `cmax`, and `error_limit` are ignored when the parameter type is set to `distr`.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.

Arguments

`type=range|distr|print`	Type of report requested.
`cmin`	Lower bound of capacitor value to be reported. Default is `-1F`.
`cmax`	Upper bound of capacitor value to be reported. Default is `1F`.

`error_limit`	Maximum number of errors to be reported when `type=print` or `type=range`. Default is 10000.

Example

Spectre Syntax

```
chk1 static_capacitor type=distr
```

SPICE Syntax

```
.cck chk1 static_capacitor type=distr
```

The simulator reports a capacitor distribution based on their value, as shown below.

Static Capacitor Check Violations

static_capacitor: chk1

chk1 static_capacitor type=distr

Violation Count: 9

Static Capacitor Check - Capacitor Value Distribution

Title	Range	Count
chk1	(-Inf, 0)	0
chk1	[0, 10 a)	1
chk1	[10 a, 100 a)	1
chk1	[100 a, 1 f)	1
chk1	[1 f, 10 f)	1
chk1	[10 f, 100 f)	1
chk1	[100 f, 1 p)	1
chk1	[1 p, 10 p)	1
chk1	[10 p, Inf)	5

Related Topics

Static Checks

Static Capacitor Voltage Check (static_capv)

Spectre Syntax

```
title static_capv cond=<expression> <inst=[inst1 inst2...]> vnth=<value>
     vpth=<value> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

SPICE Syntax

```
.cck title static_capv cond=<expression> <inst=[inst1 inst2...]> vnth=<value>
     vpth=<value> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

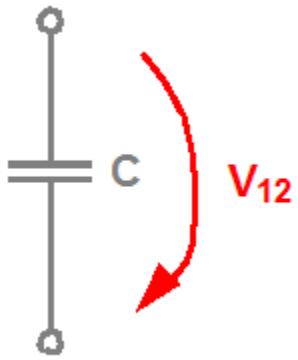
Description

Reports capacitor elements fulfilling the conditional expression on device voltages.

Supported capacitor variables are: `v(1,2)`, `v(1)`, and `v(2)`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`cond`	Condition to be checked

`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*`.nv`. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
capv1 static_capv cond="v(1,2)>0
```

Spice Syntax

```
.cck capv1 static_capv cond="v(1,2)>0
```

The above command will report the capacitor elements that fulfill the condition `v(1,2)>0`. The following is an example of the report that is displayed in the Web browser:

Static Capacitor Voltage Check Violations

static_capv: capv1

capv1 static_capv cond="v(1,2)>0"

Violation Count: 1

Static Capacitor Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	V1	V2
capv1	c1	capacitor	-	(0, 3.3)	(0, 3.3)

Related Topics

Static Checks

Static Coupling Impact Check (static_coupling)

Spectre Syntax

```
title static_coupling node=[n1 n2 ...] vnth=<value> vpth=<value>
     <inst=[inst1inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     error_limit=<value>
```

SPICE Syntax

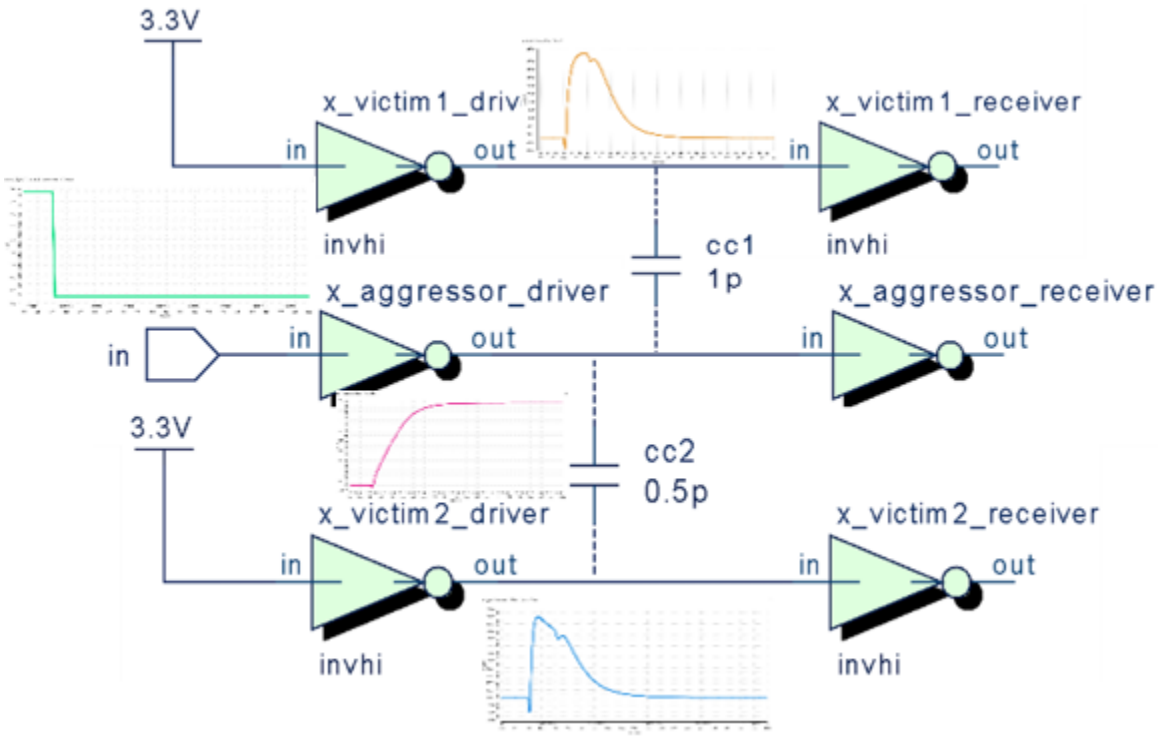
```
.cck title static_coupling node=[n1 n2 ...] vnth=<value> vpth=<value>
     <inst=[inst1inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     error_limit=<value>
```

Description

Evaluates the possible coupling effects in the circuit.

For each victim node, the check identifies its coupling impact from aggressor(s) by dividing the total charge of the aggressor with its node capacitance (including cc+gc+device cap). The charge of each aggressor is calculated by multiplying the voltage level of the coupling

aggressors (using Vmax for worst case analysis) with the coupling capacitance. In this check, only the capacitors written in netlist are counted.



The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`error_limit`	Maximum number of errors reported. Default is 100.
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation. Default is -0.4 V.
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across an NMOS channel during voltage propagation. Default is 0.5 V.
`pwl_time`	Time for pwl source to be considered as constant vsource.
`inst`	Subcircuit instances to which the check is applied. Default includes all instances (inst=*).

`node`	Nodes to which the check is applied. Default is none.
`xinst`	Subcircuit instances to be excluded from the check. Default is none.
`subckt`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
`xsubckt`	The instances of the specified subcircuits that are excluded from the check. Default is none.
`depth`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Example

Spectre Syntax

```
coup static_coupling node=[*]
```

SPICE Syntax

```
.cck coup static_coupling node=[*]
```

The above statement reports all victims that are impacted by aggressors. The following is an example of the report that is displayed in the Web browser:

Static Coupling Impact Check - Violation

Title	Node Name	Node Cap(F)	Node Min Voltage(V)	Node Max Voltage(V)	Coupling Impact
coup	victim1	1.979802e-13	0.000000e+00	0.000000e+00	1.666834e+01

Coupling_node datas: size=1

Coupling Node Name	Coupling Node Min Voltage(V)	Coupling Node Max Voltage(V)	Coupling Instance	Coupling Instance Cap(F)
aggressor	0.000000e+00	3.300000e+00	cc1	1.000000e-12

This report has the following two sections:

- The first section displays the node name of the victim. Next, the node capacitance (with respect to ground) of the victim is shown. The min/max voltage columns display the possible voltage range for this node. The coupling impact is a gauge used to measure the coupling impact of aggressor(s) on the victim. The coupling impact is a relative term and does not have any physical meaning.

- The second section displays the list of aggressors. The min/max voltage columns display the possible voltage range for this node. The coupling instance name (and its capacitance) is the capacitor connected between the aggressor and the victim. This section can have multiple rows for multiple aggressors.

Related Topics

Static Checks

Static DC Leakage Path Check (static_dcpath)

Spectre Syntax

```
title static_dcpath net=[n1 n2 ...] vnth=<value> vpth=<value> pwl_time=<value>
     error_limit=<value> <rpt_node=no|all|top|selected>
```

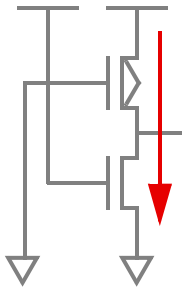
SPICE Syntax

```
.cck title static_dcpath net=[n1 n2 ...] vnth=<value> vpth=<value>
     pwl_time=<value> error_limit=<value> <rpt_node=no|all|top|selected>
```

Description

Reports the always conducting paths between the power supply nodes. If more than two nets are specified, Spectre checks the leakage path between each net combination. For example, if `net=[vdc1 vdc2 0]` is specified, then the conducting path between `vdc1` and `vdc2`, `vdc1` and `0`, and `vdc2` and `0` is checked.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`net`	Hierarchical node names between which the leakage path is checked. Wildcarding is not allowed. All combination of nets are checked. Default is none.
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).

`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*.nv. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
dc1 static_dcpath net=["vdd 0"]
```

SPICE Syntax

```
dc1 static_dcpath net=["vdd 0"]
```

The above command will report potential DC leakage paths between the power supply nodes `vdd` and `0`. The following is an example of the report that is displayed in the Web browser:

Static DC Leakage Path Check Violations

static_dcpath: dc1

dc1 static_dcpath net=["vdd" "0"]

Violation Count: 1

Static DC Leakage Path Check - Violation

Title	From Net	To Net
dc1	vdd	0

Path Elements:

mp1

mn2

Related Topics

Static Checks

Static Diode Voltage Check (static_diodev)

Spectre Syntax

```
title static_diodev model=[m1 m2 ...] cond=<expression> vnth=<value> vpth=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

SPICE Syntax

```
.cck title static_diodev model=[m1 m2 ...] cond=<expression> vnth=<value>
     vpth=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>
```

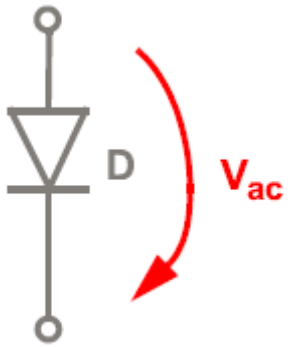
Description

Reports diode elements fulfilling the conditional expression on device voltages.

Supported diode variables are: `v(a,c)`, `v(a)`, and `v(c)`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`model`	Diode device model names to be checked.

`cond`	Condition to be checked (default is `none`).
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. depth=3 reports errors on top level and three sublevels. Default is `8`.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*.nv. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
dv1 static_diodev model=["diode1"] cond="v(a,c)>0"
```

SPICE Syntax

```
.cck dv1 static_diodev model=["diode1"] cond="v(a,c)>0"
```

The above command will report the instances of `diode1` that fulfill the condition `v(a,c)>0`. The following is an example of the report that is displayed in the Web browser:

Static Diode Voltage Check Violations

static_diodev: dv1

dv1 static_diodev model=["diode1"] cond="v(a,c)>=0"

Violation Count: 2

Static Diode Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	Va	Vc
dv1	x1.d1	diode1	inv_esd	(0.00 , 0.00)	(-5.00 , 5.00)
dv1	x1.d2	diode1	inv_esd	(-5.00 , 5.00)	(3.30 , 3.30)

Related Topics

Static Checks

Static ERC Check (static_erc)

Spectre Syntax

```
title static_erc hotwell=[off|on] dangle=[off|all|no_top]
     floatgate=[off|all|no_top|no_moscap|no_top_moscap|pode] <gate2power=off|on>
     <gate2ground=on|off> floatbulk=[off|all|no_top <inst=[inst1 inst2...]>
     vnth=<value> vpth=<value> vlth=<value> vhth=<value> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     rmax=<value> error_limit=<value> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected>
```

SPICE Syntax

```
.cck title static_erc hotwell=[off|on] dangle=[off|all|no_top]
     floatgate=[off|all|no_top|no_moscap|no_top_moscap|pode] <gate2power=off|on>
     <gate2ground=on|off> floatbulk=[off|all|no_top <inst=[inst1 inst2...]>
     vnth=<value> vpth=<value> vlth=<value> vhth=<value> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     rmax=<value> error_limit=<value> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected>
```

Description

Enables you to detect the following electrical design rule violations without running the simulation:

- MOSFET with bulk not hard-wired to power supply.
- Unconnected MOSFET gate.
- Unconnected MOSFET bulk.
- Dangling node.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.

Arguments

`hotwell`	Report MOSFET with bulk not connected to VDD or GND. Possible values are `off` and `on`.

`dangle`	Report dangling nodes. If set to `off` (default), dangling nodes are not checked. If set to `all`, all nodes are checked. If set to `no_top`, the top-level nodes are excluded from the check.
`floatgate`	Report unconnected MOSFET gates. If set to `off` (default), no nodes are checked. If set to `all`, all nodes are checked. If set to `no_top`, top-level nodes are excluded from the check. If set to `no_moscap`, MOSCAP gates are excluded from the check. If set to `no_top_moscap`, all top-level nodes and the MOSCAP gates are excluded from the check. If set to `pode`, all pode devices whose gates are floating are reported as violations. Possible values are `off`, `all`, `no_top`, `no_moscap`, `no_top_moscap`, and `pode`.
`floatbulk`	Report unconnected MOSFET bulk. If set to `off` (default), nodes are not checked. If set to `all`, all nodes are checked. If set to `no_top`, the top-level nodes are excluded from the check.
`gate2power`	Reports PMOS with gate connected to ground and NMOS with gate connected to VDD. Possible values are `on` and `off`. The default value is `off`.
`gate2ground`	Reports NMOS gate nodes connected to ground and PMOS gate nodes connected to VDD. The gate must be a vsource node. Possible values are `on` and `off`. The default value is `off`.
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`vlth`	Voltage below `vlth` is considered GND. Applicable only with `hotwell`, `gate2ground`, and `gate2power`. Default is `0.2V`.
`vhth`	Voltage above vhth is considered VDD. Applicable only with `hotwell`, `gate2ground`, and `gate2power`. Default value is `0.8V`.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.

`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`rmax`	The maximum resistance value where a node is still considered connected to voltage source node. Default is 100M.
`error_limit`	Maximum number of errors to be reported per check. Default is 10000.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named `<`*`netlist_name`*`>.<`*`check_name`*`>.nv`. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.

Example

Spectre Syntax

```
chk1 static_erc floatgate=all
```

SPICE Syntax

```
.cck chk1 static_erc floatgate=all
```

The above command reports all MOSFETs with a floating gate.

The following is an example of the report that is displayed in the Web browser:

Static ERC Check Violations

static_erc: chk1

chk1 static_erc floatgate=all

Violation Count: 2

Static ERC Check - Floating Gate Violations

Title	Instance Name
chk1	MP1
chk1	Q1.MP1

Related Topics

Static Checks

Static High Impedance Node Check (static_highz)

Spectre Syntax

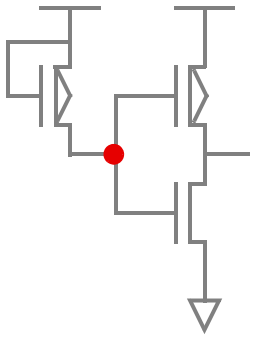
```
title static_highz  node=[n1 n2 ...] vnth=<value> vpth=<value> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <fanout=all|gate|bulk> pwl_time=<value>
     <depth=n> error_limit=<value> <rpt_node=no|all|top|selected>
```

SPICE Syntax

```
.cck title static_highz  node=[n1 n2 ...] vnth=<value> vpth=<value> <inst=[inst1
     inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> <fanout=all|gate|bulk> pwl_time=<value>
     <depth=n> error_limit=<value> <rpt_node=no|all|top|selected>
```

Description

Reports nodes without a possible conducting path to a DC power supply or ground. The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`node`	Nodes to be checked for the highz state. Default is none.
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).

`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`fanout`	Fanout setting to filter node with specified connection. Possible values are `all`, `gate`, and `bulk`. Default is `all`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*`.nv`. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
static_hz1 static_highz node=["*"]
```

SPICE Syntax

```
static_hz1 static_highz node=["*"]
```

The above command will report all possible high impedance nodes. The following is an example of the report that is displayed in the Web browser:

Static HighZ Node Check Violations

static_highz: static_hz1

static_hz1 static_highz node=["*"]

Violation Count: 1

Static HighZ Node Check - Violation

Title	Node Name
static_hz1	inp

Related Topics

Static Checks

Static Highfanout Check (static_highfanout)

Spectre Syntax

```
title static_highfanout node=[n1 n2 ...] upth=<value> downth=<value> rcut=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> <depth=n> error_limit=<value>
```

SPICE Syntax

```
.cck title static_highfanout node=[n1 n2 ...] upth=<value> downth=<value>
     rcut=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> <depth=n>
     error_limit=<value>
```

Description

The `static_highfanout` check detects MOSFETs that have a gate connected to a high fanout node. The check reports nodes that have a count greater than the specified count.

The results are written to the `static`.xml file, which can be viewed in a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`upth`	Defines the ratio of the width of PMOS (pull-up) and the loading of MOSFETs (with gate connection). The load of transmission gate is also considered. The default value is 10.
`downth`	Defines the ratio of the width of NMOS (pull-down) and the loading of MOSFETs The default value is 20.
`rcut`	Cuts the resistor having value greater than `rcut`. The default value is 1e6.
`error_limit`	Maximum number of errors reported. Default is 10000.
`inst`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
`xinst`	Subcircuit instances to be excluded from the check. Default is none.
`subckt`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).

`xsubckt`	The instances of the specified subcircuits that are excluded from the check. Default is none.
`depth`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.

Example

Spectre Syntax

```
chk1 static_highfanout node=["*"] upth=10 downth=20
```

SPICE Syntax

```
.cck chk1 static_highfanout node=["*"] upth=10 downth=20
```

The above statement reports pMOSFETs whose loading-width to driving-width ratio is more than 10. In addition, nMOSFETs whose loading-width to driving-width ratio is more than 20 are reported. The report includes the violation type, displaying if it is a pull-up violation, pull-down violation, or both.

The following is an example of the report that is displayed in the Web browser:

Static Highfanout Check Violations

static_highfanout: chk1

- chk1 static_highfanout node=["*"] upth=10 downth=20
- Violation Count: 3

Static Highfanout Node Check - Violation

Title	Node Name	UpDrivesWidth	DownDrivesWidth	LoadsWidth	UpThreshold	DownThreshold	ViolationType
chk1	x2.out	4.800000e-05	4.800000e-05	1.800000e-03	10	20	up and down
chk1	x3.inn1	3.000000e-06	0.000000e+00	9.000000e-05	10	20	up
chk1	x3.inn2	0.000000e+00	3.000000e-06	9.000000e-05	10	20	down

Related Topics

Static Checks

Static MOSFET Voltage Check (static_mosv)

Spectre Syntax

```
title static_mosv model=[m1 m2 ...] cond=<expression> vnth=<value> vpth=<value>
     <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]> <subckt=[subckt1
     subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

SPICE Syntax

```
.cck title static_mosv model=[m1 m2 ...] cond=<expression> vnth=<value>
     vpth=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>
```

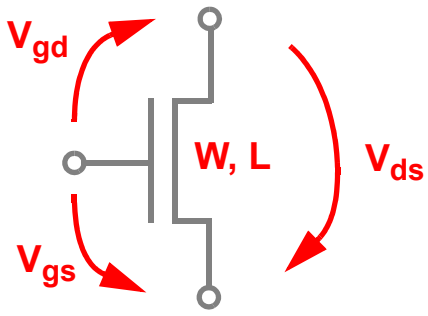
Description

Reports MOSFET devices fulfilling the conditional expression on device voltages and device size (`w`, `l`).

Supported MOSFET variables are: `v(g,s)`, `v(g,d)`, `v(g,b)`, `v(d,s)`, `v(d,b)`, `v(s,b)`, `v(g)`, `v(d)`, `v(s)`, `v(b)`, `l`, and `w`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`model`	MOSFET device model names to be checked.
`cond`	Condition to be checked
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*.nv. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
mos1 static_mosv model="nmos" cond="v(g,s)>1.9"
```

SPICE Syntax

```
.cck mos1 static_mosv model="nmos" cond="v(g,s)>1.9"
```

The above command will report the instances of `nmos` that fulfill the condition `v(g,s)>1.9`.

The following is an example of the report that is displayed in the Web browser:

Static MOSFET Voltage Check Violations

static_mosv: mos1

mos1 static_mosv model=["nmos"] cond="v(g,s)>1.9"

Violation Count: 2

Static MOSFET Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
mos1	x1.mn2	nmos	invhi	(0, 3.3)	(0, 3.3)	(0, 0)	(0, 0)
mos1	x2.mn2	nmos	inv	(0, 1.8)	(0, 3.3)	(0, 0)	(0, 0)

Related Topics

Static Checks

Static NMOS to VDD Count Check (static_nmos2vdd)

Spectre Syntax

```
title static_nmos2vdd node=[n1 n2 ...] count=<value> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> <depth=n> error_limit=<value>
```

SPICE Syntax

```
.cck title static_nmos2vdd node=[n1 n2 ...] count=<value> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> <depth=n> error_limit=<value>
```

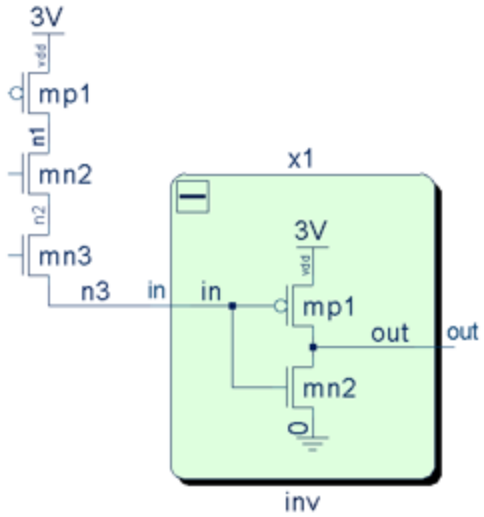
Description

Counts unpaired NMOS in the charging path from a fanout node to VDD and reports paths having NMOS count greater than the specified count

The check only analyzes the fanout nodes. It does not check the nodes connecting only to MOSDIODEs or MOSCAPs.

The check counts NMOS from a fanout node to VDD. It does not include NMOS in the count under the following conditions but does report them.

- it is connected to vdd
- it is connected parallel to PMOS as in a transmission gate



The results are written to the `static`.xml file, which can be viewed in a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`count`	Maximum number of permitted NMOS. Default is 3.
`inst`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
`xinst`	Subcircuit instances to be excluded from the check. Default is none.
`subckt`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
`xsubckt`	The instances of the specified subcircuits that are excluded from the check. Default is none.
`depth`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.
`error_limit`	Maximum number of errors reported. Default is 10000.

Example

Spectre Syntax

```
chk1 static_nmos2vdd node=[*] count=1
```

SPICE Syntax

```
.cck chk1 static_nmos2vdd node=[*] count=1
```

The above statement reports all paths from all fanout nodes having NMOS count greater than 1. The following is an example of the report that is displayed in the Web browser:

- chk1 static_nmos2vdd node=["*"] count=1
- Violation Count: 1

Static NMOS to vdd Count

Title	Node Name	Count
chk1	n3	2

Drivers datas: size=3

Driver
mn3
mn2
mp1

It reports one path from fanout `n3` to vdd. The number of NMOS found in this path is two. Note that `mp1` is a PMOS therefore does not contribute in the count.

Related Topics

Static Checks

Static NMOS/PMOS Forward Bias Bulk Check (static_nmosb/ static_pmosb)

Spectre Syntax

```
title static_nmosb model=[m1 m2 ...] mode=[definite|possible] vt=<value>
     vnth=<value> vpth=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>

title static_pmosb model=[m1 m2 ...] mode=[definite|possible] vt=<value>
     vnth=<value> vpth=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>
```

SPICE Syntax

```
.cck title static_nmosb model=[m1 m2 ...] mode=[definite|possible] vt=<value>
     vnth=<value> vpth=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>

.cck title static_pmosb model=[m1 m2 ...] mode=[definite|possible] vt=<value>
     vnth=<value> vpth=<value> <inst=[inst1 inst2...]> <xinst=[xinst1 xinst2...]>
     <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>
```

Description

Reports MOSFET devices with forward biased bulk condition. The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser. A violation is generated when the bulk bias voltage meets the following conditions:

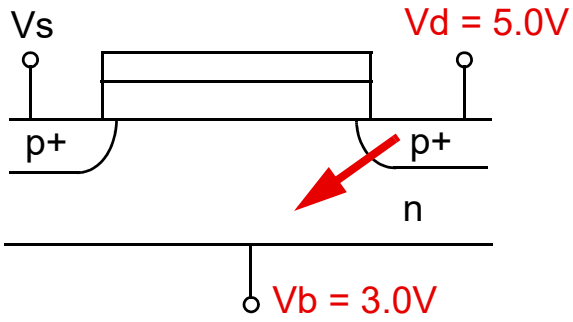
For NMOS:

- When `mode=definite`: `min( Vb ) >= min( Vd, Vs ) + abs( vt )`
- When `mode=possible`: `max( Vb ) >= min( Vd, Vs ) + abs( vt )`

For PMOS:

- When `mode=definite`: `max( Vb ) <= max( Vd, Vs ) - abs( vt )`

- When `mode=possible`: `min( Vb ) <= max( Vd, Vs ) - abs( vt )`



Arguments

`model`	Model(s) to which the forward bias bulk condition is applied. Default is `none`.
`mode`	When `mode=possible`, all possible violations are reported. When `mode=definite`, definite violations are reported. The default value is `definite`.
`vt`	Threshold voltage for p-n junction being checked (the default value is 0.3 v)
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.

`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*`.nv`. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

```
nmosb1 static_nmosb model=["nmos"]
```

SPICE Syntax

```
.cck nmosb1 static_nmosb model=["nmos"]
```

The above command will report all instances of the `nmos` model with potential forward bias bulk conditions. The following is an example of the report that is displayed in the Web browser:

Static Forward Bias Bulk Check Violations

static_nmosb: nmosb1

nmosb1 static_nmosb model=["nmos"]

Violation Count: 1

Static Forward Bias Bulk Check - Violation

Title	Instance Name	Model Name	Sub Name	Vd	Vg	Vs	Vb
nmosb1	x1.mn1	nmos	inv	(0, 3)	(0, 3)	(0, 0)	(3, 3)

Related Topics

Static Checks

Static PMOS to GND Count Check (static_pmos2gnd)

Spectre Syntax

```
title static_pmos2gnd node=[n1 n2 ...] count=<value> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> <depth=n> error_limit=<value>
```

SPICE Syntax

```
.cck title static_pmos2gnd node=[n1 n2 ...] count=<value> <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> <depth=n> error_limit=<value>
```

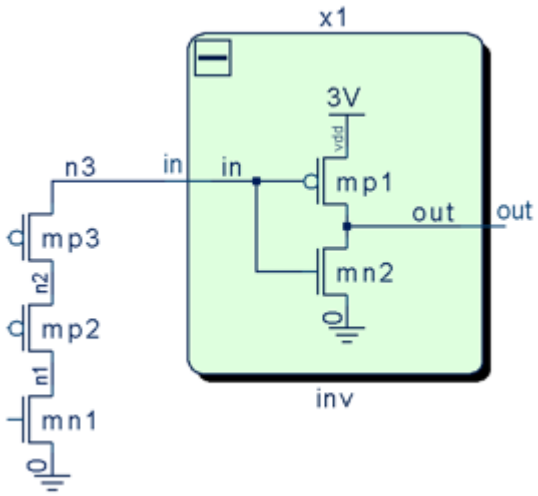
Description

Counts unpaired PMOS in the discharging path from a fanout node to GND and reports paths having PMOS count greater than the specified count.

The check only analyzes the fanout nodes. It does not check the nodes connecting only to MOSDIODEs or MOSCAPs.

The check counts PMOS from a fanout node to gnd. It does not include PMOS in the count under the following conditions but does report them.

- it is connected to gnd
- it is connected parallel to NMOS as in a transmission gate



The results are written to the `static`.xml file, which can be viewed in a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`count`	Maximum number of permitted NMOS. Default is 3.
`inst`	Subcircuit instances to which the check is applied. Default includes all instances (`inst=*`).
`xinst`	Subcircuit instances to be excluded from the check. Default is none.
`subckt`	The instances of the specified subcircuit to which the check is applied. Default includes all subcircuits (subckt=*).
`xsubckt`	The instances of the specified subcircuits that are excluded from the check. Default is none.
`depth`	Hierarchy levels (starting from top, instance, or subcircuit scope) to be checked. Default is 8.
`error_limit`	Maximum number of errors reported. Default is 10000.

Example

Spectre Syntax

```
chk1 static_pmos2gnd node=[*] count=0
```

SPICE Syntax

```
.cck chk1 static_pmos2gnd node=[*] count=0
```

The above statement reports all paths from all fanout nodes having PMOS count greater than 0. The following is an example of the report that is displayed in the Web browser:

- chk1 static_pmos2gnd node=["*"] count=0
- Violation Count: 1

Static PMOS to gnd Count

Title	Node Name	Count
chk1	n3	2

Drivers datas: size=3

Driver
mp3
mp2
mn1

It reports one path from fanout `n3` to `gnd`. The number of PMOS found in this path is two. Note that `mn1` is an NMOS therefore does not contribute in the count.

Related Topics

Static Checks

Static RC Delay Check (static_rcdelay)

Spectre Syntax

```
title static_rcdelay node=[n1 n2 ...] maxnrise=<value> minnrise=<value>
     maxnfall=<value> minnfall=<value> maxtrise=<value> mintrise=<value>
     maxtfall=<value> mintfall=<value> <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <depth=n> fanoutmargin=[lower_margin higher margin] cmin=<value>
     <detailed_path=[yes|no]>
```

SPICE Syntax

```
.cck title static_rcdelay node=[n1 n2 ...] maxnrise=<value> minnrise=<value>
     maxnfall=<value> minnfall=<value> maxtrise=<value> mintrise=<value>
     maxtfall=<value> mintfall=<value> <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <depth=n> fanoutmargin=[lower_margin higher margin] cmin=<value>
     <detailed_path=[yes|no]>
```

Description

Reports nodes with excessive rise or fall times. Rise and fall times are based on estimation.

The `static_rcdelay` check analyzes only the fanout nodes. It does not check nodes that connect to MOSDIODEs or MOSCAPs. The check reports either the top worst case rise/fall times (`maxnrise`, `maxnfall`) or nodes with rise/fall times above the user-defined thresholds (`maxtrise`, `maxtfall`). In addition, the check reports the driving MOSFETs and the receiving MOSFET (gate connected to the analyzed node) for each node.

For postlayout netlist, the check supports only the backannotation (stitching) flow.

The results are reported into a file with the extension `static.xml`, which can be read with a Web browser.

Arguments

`node`	Nodes to which the check is applied. Default is none.
`maxnrise`	Report the top number nodes with highest rise time. Default is `none`.
`minnrise`	Report the top number nodes with lowest rise time. Default is `none`.

`maxnfall`	Report the top number nodes with highest fall time. Default is `none`.
`minnfall`	Report the top number nodes with lowest fall time. Default is `none`.
`maxtrise`	Report only those node names with rise time higher than the specified value. Default is infinity.
`mintrise`	Report only those node names with rise time lower than the specified value. Default is none.
`maxtfall`	Report only those node names with fall time higher than the specified value. Default is infinity.
`mintfall`	Report only those node names with fall time lower than the specified value. Default is none.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (starting from top, instance, or subcircuit scope) to be checked. Default is 8.
`fanoutmargin`	Relative fanout level (in ratio of VDD voltage) for which rise and fall time is measured. The range of values is `[0.01 0.99]`. The default range is `[0.1 0.9]`. Lower margin should be less than the higher margin.
`cmin`	Node capacitance threshold. Only nodes with total capacitance higher than the specified value are reported. Default is 1e-14.
`detailed_path`	Report all possible (yes) or worst case (no) rise/fall time per node.Possible values are `yes` and `no`.

Example

Spectre Syntax

```
chk static_rcdelay node=[*] maxnrise=10 cmin=1e-14 detailed_path=no
```

SPICE Syntax

```
.cck chk static_rcdelay node=[*] maxnrise=10 cmin=1e-14 detailed_path=no
```

The above command will estimate the RC delay for all nodes. It will report four tables, one table for each parameter. It will report the maximum (highest) two rise delays.

The following is an example of the report that is displayed in the Web browser:

static_rcdelay: chk

- chk static_rcdelay node=["*"] maxnrise=10 cmin=1e-14 detailed_path=no
- Violation Count: 4

Static RC Delay Check - Max Rise-time Delays

Title	Node Name	Delay(s)	Receiver	Node Cap(F)
chk	E	1.217681e-09	x_d2.mp2	2.029682e-13

Drivers datas: size=2

Driver
x_r1.mp2
x_r1.mp1

Related Topics

Static Checks

Static Resistor Check (static_resistor)

Spectre Syntax

```
title static_resistor type=[range|distr|print] rmin=<value> rmax=<value>
     error_limit=<value>
```

SPICE Syntax

```
.cck title static_resistor type=[range|distr|print] rmin=<value> rmax=<value>
     error_limit=<value>
```

Description

Reports all resistors within or outside the range `rmin` and `rmax`. Moreover, it also generates a distribution list of all resistors in the circuit.

If the parameter type is set to `range`, then all resistors outside the range of `rmin` and `rmax` will be reported.

If the parameter type is set to `print`, then all resistors between `rmin` and `rmax` will be reported. The resistor names can be sorted by clicking the *Device name* header.

If the parameter type is set to `distr` then a distribution report will be generated for all resistors. There are 12 bins that are:

`-Inf - 0, 0 - 1m, 1m - 10m, 10m - 0.1, 0.1 - 1, 1 - 10, 10 - 100, 100 - 1k, 1k - 10k, 10k - 100k, 100k - 1Meg, and 1Meg - Inf`

However, if two or more consecutive bins are empty then they will merge into one bin, reducing the number of bins. The parameters `rmin`, `rmax` and `error_limit` are ignored when the parameter type is set to `distr`.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.

Arguments

`type=range|distr|print`	Type of report requested.
`rmin`	Lower bound of resistor value to be reported. Default is `-1000G` Ohm.

`rmax`	Upper bound of resistor value to be reported. Default is `1000G` Ohm.
`error_limit`	Maximum number of errors to be reported when `type=print` or `type=range`. Default is 10000.

Example

Spectre Syntax

```
chk1 static_resistor type=distr
```

SPICE Syntax

```
.cck chk1 static_resistor type=distr
```

The simulator generates a resistor value distribution report, as shown below.

Static Resistor Check Violations

static_resistor: chk1

chk1 static_resistor type=distr

Violation Count: 12

Static Resistor Check - Resistor Value Distribution

Title	Range	Count
chk1	(-Inf, 0)	0
chk1	[0, 1 m)	1
chk1	[1 m, 10 m)	1
chk1	[10 m, 100 m)	1
chk1	[100 m, 1)	1
chk1	[1 , 10)	1
chk1	[10 , 100)	1
chk1	[100 , 1 K)	1
chk1	[1 K, 10 K)	1
chk1	[10 K, 100 K)	1
chk1	[100 K, 1 M)	1
chk1	[1 M, Inf)	2

Related Topics

Static Checks

Static Resistor Voltage Check (static_resv)

Spectre Syntax

```
title static_resv cond=<expression> <inst=[inst1 inst2...]> vnth=<value>
     vpth=<value> <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]>
     <xsubckt=[xsubckt1 xsubckt2....]> pwl_time=<value> <depth=n>
     <rpt_node=no|all|top|selected> error_limit=<value>
```

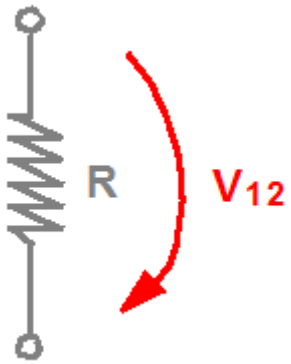
Description

Reports resistor elements fulfilling the conditional expression on device voltages.

Supported resistor variables are: `v(1,2)`, `v(1)`, and `v(2)`.

Supported operators are: +, -,*, /, ==,!=, <, <=, >, >=,||, &&, and !.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`cond`	Condition to be checked
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across a NMOS channel during voltage propagation (the default value is 0.5 v).

`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (the default value is -0.4 v).
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*`.nv`. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

```
resv1 static_resv cond="v(1,2)>0
```

The above command will report the resistor elements that fulfill the condition `v(1,2)>0`.

The following is an example of the report that is displayed in the Web browser:

Static Resistor Voltage Check Violations

static_resv: resv1

resv1 static_resv cond="v(1,2)>0"

Violation Count: 1

Static Resistor Voltage Check - Violation

Title	Instance Name	Model Name	Sub Name	V1	V2
resv1	r1	resistor	-	(0, 3.3)	(0, 3.3)

Related Topics

Static Checks

Static Subcircuit Port Voltage Check (static_subcktport)

Spectre Syntax

```
title static_subcktport cond=<expression> vnth=<value> vpth=<value> subckt=<value>
     pwl_time=<value> error_limit=<value>
```

SPICE Syntax

```
.cck title static_subcktport cond=<expression> vnth=<value> vpth=<value>
     subckt=<value> pwl_time=<value> error_limit=<value>
```

Description

Reports instances of the user-specified `subckt` fulfilling the conditional expression on port voltages. Only one subcircuit is allowed per statement.

Voltages of a port can be referenced by the port name of a subcircuit. For example, consider the subcircuit definition `.subckt INV port_A port_B`. Here, supported port names are: `v(port_A)`, `v(port_B)` and `v(port_A,port_B)`.

Supported operators are: `+`, `-`,`*`, `/`, `==`,`!=`, `<`, `<=`, `>`, `>=`,`||`, `&&`, and `!`.

All design instances of the subcircuit, specified using `subckt` are checked.

The results are written to the `static`.xml file, which can be viewed in a Web browser.

Arguments

`cond`	Conditional expression to be fulfilled. Default is none.
`vnth`	NMOS threshold voltage. This value is used to calculate the voltage drop across an NMOS channel during voltage propagation (default value is `0.5 v`).
`vpth`	PMOS threshold voltage. This value is used to calculate the voltage drop across a PMOS channel during voltage propagation (default value is `-0.4 v`).
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`error_limit`	Maximum number of errors reported. Default is 10000.

`subckt`	Instances of the specified subcircuit to which the check is applied. Wildcard is not supported. Default is none.

Example

Spectre Syntax

```
sp1 static_subcktport subckt="decand3_pre" cond="v(Z)>1.7"
```

SPICE Syntax

```
.cck sp1 static_subcktport subckt="decand3_pre" cond="v(Z)>1.7"
```

The above command will report all instances satisfying the condition `v(Z)>1.7`. The following is an example of the report that is displayed in the Web browser:

static_subcktport: sp1

- sp1 static_subcktport subckt="decand3_pre" cond="v(Z)>1.7"
- Violation Count: 16

Static Subckt Port Check - Violation

Title	Inst Name	Key Sub-expressions	Operands Value
sp1	XTOP.XPRE1.XI2	(v(Z)>1.7)	v(Z)=(0, 1.8)
sp1	XTOP.XPRE1.XI7	(v(Z)>1.7)	v(Z)=(0, 1.8)

Related Topics

Static Checks

Static Transmission Gate Check (static_tgate)

Spectre Syntax

```
title static_tgate node=[n1 n2 ...] <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <depth=n> error_limit=<value>
```

SPICE Syntax

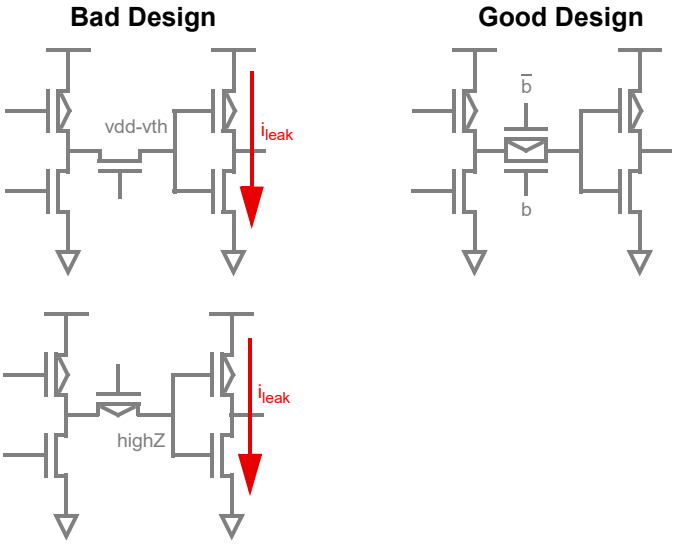
```
.cck title static_tgate node=[n1 n2 ...] <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     <depth=n> error_limit=<value>
```

Description

Reports transmission gates which may cause potential leakage currents between power supplies. Such gates can be characterized by their node connectivity, based on the following:

- Nodes which connect to gate and NMOS drain/source terminals, but not to PMOS drain/source terminals
- Nodes which connect to gate and PMOS drain/source terminals, but not to NMOS drain/source terminals

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`node`	Nodes to be checked. Default is none.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

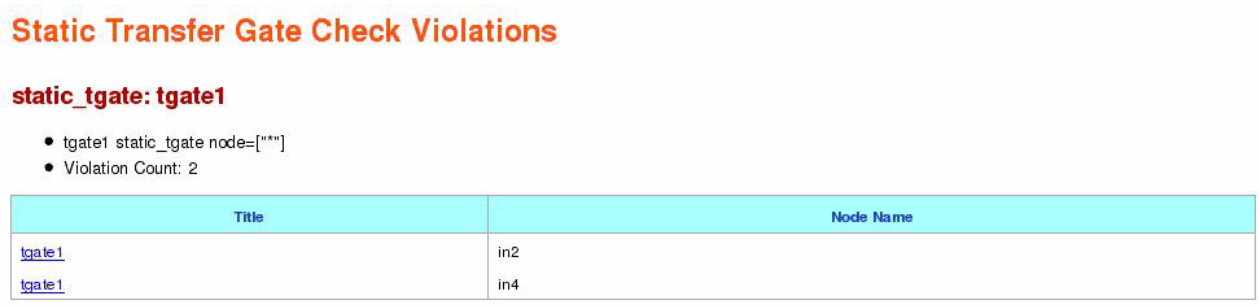
Spectre Syntax

```
tgate1 static_tgate node=["*"]
```

SPICE Syntax

```
.cck tgate1 static_tgate node=["*"]
```

The above command will report all nodes connecting to transmission gates that may cause design problems like leakage currents. The following is an example of the report that is displayed in the Web browser:



Static Transfer Gate Check Violations

static_tgate: tgate1

- tgate1 static_tgate node=["*"]
- Violation Count: 2

Title	Node Name
tgate1	in2
tgate1	in4

Related Topics

Static Checks

Static Voltage Domain Device Check (static_voltdomain)

Spectre Syntax

```
title static_voltdomain model=[m1 m2 ...] <inst=[inst1 inst2...]> <xinst=[xinst1
     xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1 xsubckt2....]>
     pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>
```

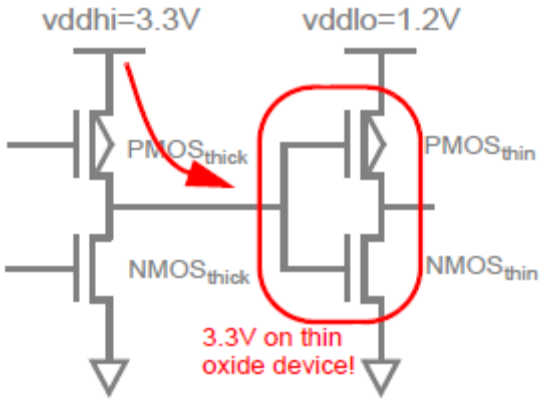
SPICE Syntax

```
.cck title static_voltdomain model=[m1 m2 ...] <inst=[inst1 inst2...]>
     <xinst=[xinst1 xinst2...]> <subckt=[subckt1 subckt2....]> <xsubckt=[xsubckt1
     xsubckt2....]> pwl_time=<value> <depth=n> <rpt_node=no|all|top|selected>
     error_limit=<value>
```

Description

Reports high voltage driving the low-voltage MOSFETS and low voltage driving the high-voltage MOSFETs.

The results are written to a file with the extension `static.xml`, which can be viewed with a Web browser.



Arguments

`model`	MOSFET device model names to be checked. By default, all types of MOSFETs are checked.
`inst`	Subcircuit instances to which the circuit check is applied. Default is `inst=*`
`xinst`	Subcircuit instances that are excluded from the circuit check. Default is `none`.
`subckt`	Instances of the specified subcircuit to which the circuit checks are applied. Default is `subckt=*`.
`xsubckt`	Instances of the specified subcircuit which are excluded from the circuit checks. Default is `none`.
`pwl_time`	If specified, all pwl sources are considered constant sources with a value equal to the voltage of the pwl source at `pwl_time`. Default is infinity.
`depth`	Hierarchy level (from top level) to be checked. `depth=3` reports errors on top level plus 3 sublevels. Default is 8.
`rpt_node`	Report node voltages. If set to `no`, no node voltages are reported. If set to `all`, all node voltages are reported. If set to `top`, voltages of only the top-level nodes are reported. If set to `selected`, voltages of only the specified nodes are reported. The report is written to a file named *<netlist_name>.<check_name>*.nv. Possible values are `no`, `all`, `top`, and `selected`. The default value is `no`.
`error_limit`	Maximum number of errors to be reported. Default is 10000.

Example

Spectre Syntax

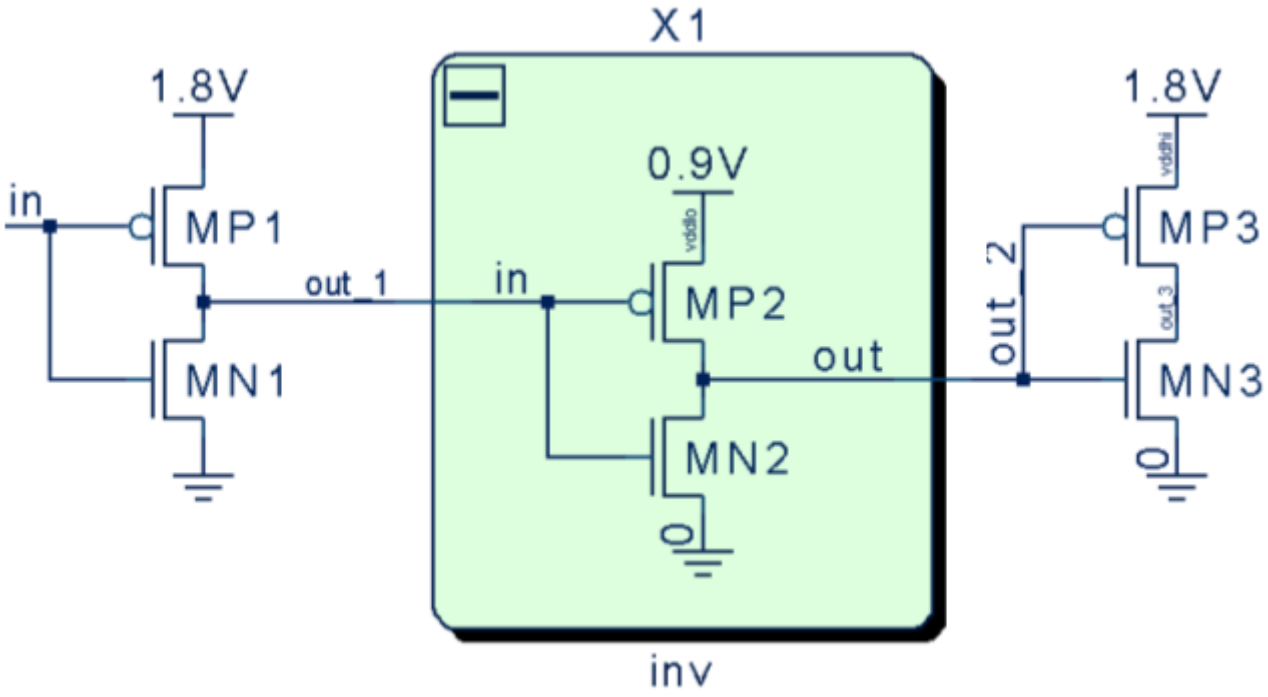
```
chk1 static_voltdomain model=[*]
```

SPICE Syntax

```
.cck chk1 static_voltdomain model=[*]
```

The above command will check all instances of MOSFETS. The *HighV Driving LowV* table reports MOSFETs whose gate voltage is higher than the drain/source voltage. The *LowV*

Driving HighV table reports MOSFETs whose gate voltage is lower than the drain/source voltage.



The following is an example of the report that is displayed in the Web browser:

Static Voltage Domain Device Check - HighV Driving LowV

Title	Gate Name	Instance Name	Subckt Name	Vd	Vg	Vs	Vb
chk1	out_1	X1.MN2	inv	(0, 0.9)	(0, 1.8)	(0, 0)	(0, 0)
chk1	out_1	X1.MP2	inv	(0, 0.9)	(0, 1.8)	(0.9, 0.9)	(0.9, 0.9)

Static Voltage Domain Device Check - LowV Driving HighV

Title	Gate Name	Instance Name	Subckt Name	Vd	Vg	Vs	Vb
chk1	in	MN1	N/A	(0, 1.8)	(0, 0.9)	(0, 0)	(0, 0)
chk1	out_2	MN3	N/A	(0, 1.8)	(0, 0.9)	(0, 0)	(0, 0)
chk1	in	MP1	N/A	(0, 1.8)	(0, 0.9)	(1.8, 1.8)	(1.8, 1.8)
chk1	out_2	MP3	N/A	(0, 1.8)	(0, 0.9)	(1.8, 1.8)	(1.8, 1.8)

Note that the *Subckt Name* is the name of subcircuit containing the instance `X1.MN2`. This helps in locating level-shifters. If an instance is at the top-level then *Subckt Name* will show `N/A`.

Related Topics

Static Checks