NanoTime
Variables and Attributes

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Copyright Notice for the Command-Line Editing Feature

Copyright Notice for the Line-Editing Library

Contents

bus_notation

Specifies the left, middle, and right characters used in bus notation.

TYPE

string

DEFAULT

[:]

DESCRIPTION

This variable specifies the characters used in bus notation. The default is [:], which means that the notation for bit 3 of a bus called ABUS can be written as ABUS[3], and the range of bits from 3 to 7 can be written as ABUS[3:7].

When NanoTime reads in a netlist that contains buses, it flattens them into individual nets. The bus notation convention affects the reading of netlists and the interpretation of commands. You can use the bus name to represent some or all bits of the bus.

For example, the following command sets the input delay to 5.0 time units for all bits of bus ABUS:

```
nt_shell> set_input_delay 5.0 [get_ports ABUS]
```

To set the input delay for only bits 0 through 4:

```
nt_shell> set_input_delay 4.0 [get_ports ABUS[0:4]]
```

To set the input delay for only bit 0:

```
nt_shell> set_input_delay 3.0 [get_ports ABUS[0]]
```

SEE ALSO

`register_netlist(2)`

ccs_enable_driver_delay_calculation

Specifies whether NanoTime uses the CCS driver models of CCS-based timing models.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable composite current source (CCS) timing delay calculation. If used, the variable must be set before the **check_design** command. You must have a NanoTime Ultra license to use CCS models.

When a CCS-based timing model is the driver of a net, this variable specifies whether NanoTime uses the CCS driver model. If the variable is set to **false** (the default), any CCS drivers in a timing model are ignored and the NLDM drivers are used instead. If the variable is set to **true**, the CCS driver models in a timing model are used if available; otherwise, the NLDM driver models are used.

SEE ALSO

`ccs_enable_receiver_delay_calculation(3)`

ccs_enable_receiver_delay_calculation

Specifies whether NanoTime uses the CCS receiver capacitance models of CCS-based timing models.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable composite current source (CCS) timing delay calculation. If used, the variable must be set before the **check_design** command.

When a CCS-based timing model is a receiver on a net, this variable specifies whether NanoTime uses the CCS receiver capacitance model. If the variable is set to **false** (the default), any CCS receiver capacitance models are ignored and the NLDM receiver capacitance models are used instead. If the variable is set to **true**, the CCS receiver capacitance models in a timing model are used if available; otherwise, the NLDM receiver capacitance models are used.

SEE ALSO

`ccs_enable_driver_delay_calculation(3)`

collection_result_display_limit

Sets the maximum number of objects that can be displayed by any command that displays a collection.

TYPE

integer

DEFAULT

100

DESCRIPTION

This variable sets the maximum number of objects that can be displayed by any command that displays a collection. The default is 100.

When a command (for example, the **add_to_collection** command) is issued at the command prompt, its result is implicitly queried, as though the **query_objects** command had been called. You can limit the number of objects displayed by setting this variable to an appropriate integer. A value of -1 displays all objects; a value of 0 displays the collection handle ID instead of the names of any objects in the collection.

SEE ALSO

collections(2)
printvar(2)
query_objects(2)

dcs_control_header

Specifies the control header file for dynamic clock simulation.

TYPE

string

DEFAULT

""

DESCRIPTION

The **dcs_control_header** variable specifies the name of the dynamic clock simulation control header file, which is included as a configuration file sent to the embedded dynamic simulation tool. The control header file can contain any valid control statements for the dynamic simulator.

SEE ALSO

dcs_enable_analysis(3)
dcs_number_of_cycles(3)
dcs_spice_header(3)
dcs_simulator(3)
dcs_enable_detailed_path_reporting(3)

dcs_enable_analysis

Enables or disables dynamic clock simulation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set the **dcs_enable_analysis** variable to **true** to enable dynamic clock simulation (DCS). You must set this variable before executing the **check_design** command. By default, dynamic clock simulation is disabled.

When dynamic clock simulation is enabled, the **check_design** command invokes the embedded simulation tool to analyze the clock network. NanoTime uses the delay and slew results from the dynamic simulation to perform timing analysis.

During dynamic clock simulation, the values of the **rc_slew_lower_threshold_pct_rise** and **rc_slew_upper_threshold_pct_rise** variables are used as the slew thresholds on all nets in the dynamically simulated clock network.

SEE ALSO

`dcs_control_header(3)`
`dcs_number_of_cycles(3)`

```
dcs_simulator(3)
dcs_spice_header(3)
dcs_enable_detailed_path_reporting(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_rise(3)
```

dcs_enable_detailed_path_reporting

Enables or disables detailed path reporting for Dynamic Clock Simulation regions.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable detailed path reporting within the dynamic clock simulation (DCS) region. The variable must be set before executing the **check_design** command.

When detailed path reporting is disabled (the default), the **report_paths** command shows the path through the clock network as a single arc from the input to the output of the dynamic clock simulation region, with no detail of the path taken within the DCS region.

When detailed path reporting is enabled, the **report_paths** command shows the critical path through the dynamic clock simulation region as a sequence of arcs traversing the internal timing points of the region, much like other path reports. Extra information is tracked and saved when detailed path reporting is enabled, resulting in increased runtime and memory requirements.

To collect the extra information, NanoTime performs path tracing within the DCS region. If path tracing fails, NanoTime might not be able to calculate clock arrival times; in this case, the tool issues an NTM-002 warning message about missing clock arrivals. Path tracing can fail for reasons such as improper timing constraints, incorrectly recognized or marked topologies, or incorrect clock net propagation. However, the dynamic clock simulation results are accurate even if the tool issues warning messages when the **dcs_enable_detailed_path_reporting** variable is **true**. If no errors are encountered when the **dcs_enable_detailed_path_reporting** variable is **false**, the DCS results are accurate.

When detailed path reporting is enabled, the **report_clock_arrivals** command generates a report that

includes clock arrivals at internal timing points within the DCS region for the critical path. However, min and max paths from source clock to sink endpoints are not guaranteed to sensitize all nodes within the DCS region. To see clock arrivals for all timing points within the DCS region, set both the **dcs_enable_detailed_path_reporting** and **dcs_enable_detailed_path_reporting_complete** variables to **true**. This combination might significantly increase runtime and memory consumption because additional simulations might be necessary to gather the information.

This feature requires a NanoTime Ultra license.

SEE ALSO

dcs_enable_analysis(3)
dcs_enable_detailed_path_reporting_complete(3)
NTM-002(n)

dcs_enable_detailed_path_reporting_complete

Enables or disables determination of clock arrivals on all internal nodes of Dynamic Clock Simulation regions.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable determination of clock arrivals on all internal nodes of Dynamic Clock Simulation (DCS) regions. This must be done before the **check_design** command. By default, the variable is set to **false**. The **dcs_enable_detailed_path_reporting** variable must also be set to **true** for this variable to take effect.

Min and max paths through the Dynamic Clock Simulation region from source clock to sink endpoints are not guaranteed to sensitize all nodes within the region. To see clock arrivals at all nodes using the **report_clock_arrivals** command, set this variable to **true**. Since many additional DCS simulations might be performed to gather this extra information, the runtime and memory consumption might increase significantly.

This feature requires a NanoTime Ultra license.

SEE ALSO

```
dcs_enable_detailed_path_reporting(3)
dcs_enable_analysis(3)
```

dcs_enable_internal_coupling_caps

Enables or disables the use of internal coupling capacitors for Dynamic Clock Simulation delay calculations.

TYPE

boolean

DEFAULT

true

DESCRIPTION

If this variable is set to true, cross-coupling capacitors that are internal to the Dynamic Clock Simulation (DCS) region will be modeled as capacitors between the signal nets. This is the most accurate setting.

To read in parasitics and retain the cross-coupling capacitors, you must enable SI analysis or noise analysis, and you must use the **-keep_capacitive_coupling** option of **read_parasitics** or **link_design**.

If this variable is set to false, cross-coupling capacitors that are internal to the Dynamic Clock Simulation (DCS) region will be split into two separate grounded capacitors. This setting is available for backwards compatibility with the previous behavior.

SEE ALSO

```
dcs_enable_analysis(3)
si_enable_analysis(3)
si_enable_noise_analysis(3)
link_design(2)
read_parasitics(2)
```

dcs_enable_network_reduction

Enables or disables reduction of the DCS network.

TYPE

boolean

DEFAULT

false

DESCRIPTION

Set this variable to true to reduce the size and scope of the Dynamic Clock Simulation (DCS) region. This must be done before the **check_design** command. By default, the variable is set to false.

When dcs_enable_analysis is set to true, the default behavior is to include the whole clock network in a single simulation. For some designs this can create a very large network which requires a long simulation runtime. When this variable is true an attempt is made to reduce the size of the DCS region. This is done by looking at all outputs of the DCS region and determining if they are driven by inverters. For outputs driven by inverters, the inverters are removed from the DCS network, and the boundary of the DCS network is moved back to the inverter inputs.

The command mark_clock_network -end_dcs <objects> can be used to manually perform this operation or to exclude stages that are not inverters.

SEE ALSO

`mark_clock_network(2)`

```
dcs_enable_analysis(3)
```

dcs_enable_si_analysis

Allows the dynamic clock simulation region to be included when SI delay analysis is enabled.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, dynamic clock simulation (DCS) includes the effects of parasitic capacitances that reside completely within the simulation region. Setting the **dcs_enable_si_analysis** variable to **false** disables this capacitance analysis.

If the **dcs_enable_si_analysis** variable is **true** and you also enable global signal integrity (SI) analysis by setting the **si_enable_analysis** variable to **true**, NanoTime includes the capacitive coupling between nets inside the DCS region and nets outside the region, resulting in more accurate delays.

To reduce runtime, only the logic considerations from the first iteration of signal integrity analysis are used as input for the dynamic clock simulation. Subsequent iterations of SI analysis reuse the same delays. You can improve delay accuracy, at the cost of runtime, by setting the **dcs_enable_si_analysis_first_iteration_only** variable to **false**.

SEE ALSO

`dcs_enable_analysis(3)`

```
dcs_enable_detailed_path_reporting(3)
dcs_enable_si_analysis_first_iteration_only(3)
si_enable_analysis(3)
```

dcs_enable_si_analysis_first_iteration_only

Specifies that only the first iteration of signal integrity analysis is used as input for dynamic clock simulation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

The **dcs_enable_si_analysis_first_iteration_only** variable controls which types of signal integrity (SI) effects are considered for the dynamic clock simulation (DCS) region when the **dcs_enable_si_analysis** variable is set to **true**.

To reduce runtime, only the logic considerations from the first iteration of signal integrity analysis are used as input for the dynamic clock simulation. Subsequent iterations of SI analysis reuse the same delays. You can improve delay accuracy, at the cost of runtime, by setting the **dcs_enable_si_analysis_first_iteration_only** variable to **false**. In this case, subsequent SI iterations consider victim-aggressor timing windows at the cost of performing more dynamic simulations.

SEE ALSO

```
dcs_enable_analysis(3)
dcs_enable_si_analysis(3)
si_enable_analysis(3)
```

```
si_enable_first_iteration_pessimism_reduction(3)
```

dcs_number_of_cycles

Specifies the number of reference clock cycles to simulate for Dynamic Clock Simulation.

TYPE

integer

DEFAULT

5

DESCRIPTION

This variable specifies the number of reference clock cycles to simulate for Dynamic Clock Simulation. This value is multiplied by the longest clock period to get the simulation time span.

The default setting is 5, which results in the simulation of five cycles of the reference clock.

If a complex clock divider needs several cycles after startup to stabilize, you might need to specify a larger number of clock cycles.

SEE ALSO

```
dcs_control_header(3)
dcs_enable_analysis(3)
dcs_spice_header(3)
dcs_enable_detailed_path_reporting(3)
```

dcs_simulator

Specifies the simulator to be used for dynamic clock simulation.

TYPE

string

DEFAULT

finesim

DESCRIPTION

This variable specifies the simulator that NanoTime uses for dynamic clock simulation (DCS). The only valid value is **finesim**, which invokes the Synopsys FineSim tool.

SEE ALSO

```
dcs_control_header(3)
dcs_enable_analysis(3)
dcs_number_of_cycles(3)
dcs_enable_detailed_path_reporting(3)
```

dcs_spice_header

Specifies the SPICE header file for use with dynamic clock simulation (DCS).

TYPE

string

DEFAULT

""

DESCRIPTION

The **dcs_spice_header** variable specifies the name of a SPICE header file to be used with dynamic clock simulation (DCS).

The contents of the header file are included at the head of the SPICE-format netlist sent to the embedded dynamic simulation tool. The header file can contain any valid SPICE control statements accepted by the simulator. Models specified with the **read_spice_models** command have higher priority.

These files specify the SPICE models and *nanosim tech commands needed to run the dynamic simulator. For example:

```
*nanosim tech="delta_vt -0.2 0.2"
*nanosim tech="voltage 1.079"
*nanosim tech="vds 0 1.3 .1"
*nanosim tech="vgs 0 1.3 .1"
*nanosim tech="vbs 0 1.3 .1"
*nanosim tech="direct"
.temp 125
.lib '/remote/myspice.lib' SS
```

If no file name is specified, NanoTime looks for a SPICE header file named nt_dcs_spice_header.cfg.

SEE ALSO

dcs_control_header(3)
dcs_enable_analysis(3)
dcs_number_of_cycles(3)
dcs_simulator(3)
dcs_enable_detailed_path_reporting(3)

dds_recalc_slew_tolerance

Specifies the maximum allowable variation of the input transition to a dynamic delay simulation region before computation of new delay values is required.

TYPE

float

DEFAULT

0.02

DESCRIPTION

The **dds_recalc_slew_tolerance** variable specifies the maximum allowed amount of variation of an input transition to a dynamic delay simulation (DDS) region for reuse of cached delay values. If this threshold is exceeded, new delay values for the simulation region are computed. This improves runtime for designs with a large number of DDS regions, at the possible cost of a small loss of accuracy for these regions.

SEE ALSO

`mark_simulation(2)`
`set_simulation_attributes(2)`

dds_simulator

Specifies the simulator to be used for dynamic delay simulation.

TYPE

string

DEFAULT

finesim

DESCRIPTION

This variable specifies the simulator that NanoTime uses for dynamic delay simulation (DDS). The only valid value is **finesim**, which invokes the Synopsys FineSim tool.

SEE ALSO

`mark_simulation(2)`
`set_simulation_attributes(2)`

default_top_name

Specifies the default top-level module name.

TYPE

string

DEFAULT

top

DESCRIPTION

This variable specifies the default top-level module name when no name is specified in the **link_design** command.

SEE ALSO

`link_design(2)`

distributed_processing_force_execute

Enables the execution of distributed processing when pre-testing fails.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime performs pre-testing for distributed processing when a host file is specified. When user environments are not compatible with the user farm, the testing can fail. In this case, the command which uses distributed processing fails by default. If this variable is set to true, NanoTime will execute the command without distributed processing.

SEE ALSO

```
extract_model(2)
distributed_processing_host_file(3)
```

distributed_processing_host_file

Specifies the host definition file that describes the host machines for distributed processing.

TYPE

string

DEFAULT

""

DESCRIPTION

NanoTime supports distributed processing in some contexts, such as the **extract_model** command. To use distributed processing, you must specify names and properties of available computing resources in a host file. The **distributed_processing_host_file** variable contains the name of the host file.

Observe the following guidelines when creating a host file:

1. Each valid host definition occupies an entire line.

2. A line starting with the pound character (#) is a comment.

3. Each line contains six fields separated by the vertical bar delimiter (|) in the following format. You must use a space for an empty field.

```
enabled | hostname | num_slots | tmpDir | mode | command
```

The fields are as follows:

- enabled: 1 means the host is enabled. Any other value indicates the host is disabled.
- hostname: This field is ignored for SGE and LSF entries.
- num_slots: The number of slots on this host (RSH and SSH) or farm (LSF and SGE). A value of -1

indicates an unlimited number.

- tmpDir: The temporary working directory. This field is optional.
- mode: RSH, SSH, LSF, SGE, SH, PBS, or RTDA
- command: The command to connect to the remote machine.

EXAMPLES

```
# Host file
1 | my_machine1 | 2 | ~/cdpl_tmp | RSH | rsh
1 | my_machine2 | 2 | ~/cdpl_tmp | RSH | rsh
```

SEE ALSO

extract_model(2)
distributed_processing_number_hspice_license(3)
distributed_processing_perl_path(3)

distributed_processing_instance_name

Read-only variable to fetch the name associated with the current NanoTime instance.

TYPE

string

DEFAULT

empty string

DESCRIPTION

This variable can be used to obtain the unique name associated with the current instance and use it to select between multiple tasks.

SEE ALSO

```
distributed_processing_host_file(3)
distributed_processing_is_master(3)
distributed_processing_result_list(3)
```

distributed_processing_is_master

Read-only variable that indicates whether the instance is a master.

TYPE

boolean

DEFAULT

false

DESCRIPTION

This variable can be used to structure the control logic in a Tcl script.

SEE ALSO

distributed_processing_host_file(3)
distributed_processing_instance_name(3)
distributed_processing_result_list(3)

distributed_processing_number_hspice_license

Specifies the number of HSPICE licenses used in distributed HSPICE simulations.

TYPE

integer

DEFAULT

0

DESCRIPTION

NanoTime supports distributed processing in some commands, such as the **extract_model** command and the **write_spice -simulate** command. When you perform HSPICE simulations in parallel, the distributed processing is limited by the number of HSPICE licenses. Use this variable to specify the maximum number of HSPICE licenses used for distributed HSPICE simulations. The default value 0 means unlimited HSPICE licenses.

SEE ALSO

```
extract_model(2)
write_spice(2)
distributed_processing_host_file(3)
```

distributed_processing_perl_path

Specifies the complete path of the Perl executable.

TYPE

string

DEFAULT

/usr/local/bin/perl

DESCRIPTION

During distributed processing, NanoTime creates Perl scripts to control the load balance between distributed jobs. Set the **distributed_processing_perl_path** variable to the full path of the Perl executable file in your computing environment.

NanoTime can use distributed processing to improve runtime for complex tasks. One example is when the **extract_model** command is executed with the **-hspice_timing** option. Another example is when parallel NanoTime runs are executed with the **run_parallel** command.

Distributed processing is controlled by setting global variables, including the following:

- For HSPICE recalibration runs, the maximum number of parallel HSPICE jobs can be specified with the **distributed_processing_number_hspice_license** variable.

- The host file must be specified by the **distributed_processing_host_file** variable.

- The Perl path must be specified by the **distributed_processing_perl_path** variable.

SEE ALSO

extract_model(2)
distributed_processing_host_file(3)
distributed_processing_number_hspice_license(3)

distributed_processing_result_list

Returns a list that encodes the result from each of the instances spawned by the most recent **run_parallel** command. A '1' indicates that the instance completed successfully, '0' otherwise.

TYPE

string

DEFAULT

Empty string

DESCRIPTION

This is a read-only variable.

To determine the current value of this variable, type the following:

printvar distributed_processing_result_list

SEE ALSO

run_parallel(2)

hierarchy_separator

Specifies the hierarchy separator character.

TYPE

string

DEFAULT

. (period character)

DESCRIPTION

This variable specifies the ASCII character used to delimit hierarchical levels when NanoTime reads in a netlist and when it generates reports with hierarchical names. By default, a period character (.) is used.

The following characters can be specified as the hierarchy separator:

```
      at sign  @
        caret  ^
   pound sign  #
       period  .
        slash  /
 vertical bar  |
```

You can set the variable either by using a dedicated command, **set_hierarchy_separator**, or by using the **set** command. For example, these two commands have the same effect:

```
nt_shell> set_hierarchy_separator #
1
nt_shell> set hierarchy_separator #
#
```

To view the current setting:

```
nt_shell> printvar hierarchy_separator
hierarchy_separator  = "#"
```

In most situations, you should use the default, the period character. However, in cases where the hierarchy character is embedded in names, a search through the design might return incorrect results. The **set_hierarchy_separator** command provides a convenient method for dealing with this situation.

SEE ALSO

set_hierarchy_separator(2)

lib_capacitance_inf

Specifies the value for library capacitance values defined as **inf**.

TYPE

float

DEFAULT

1e15

DESCRIPTION

This parameter specifies the numerical value to be used for capacitances whose value in the library data has been specified as **inf**.

This value uses the unit defined by variable **lib_capacitance_unit**. Please note that such near-infinite capacitance values can provoke fanout loads that exceed reasonable ranges for its drivers if any.

SEE ALSO

`lib_capacitance_unit(3)`

lib_capacitance_unit

Specifies the unit size for capacitance.

TYPE

string

DEFAULT

1pf

DESCRIPTION

This parameter specifies the capacitance units used in the design (for example, 1pf for picofarads or 1ff for femtofarads). NanoTime places the specified unit size into the built-in element library when you link the design with the **link_design** command. When NanoTime reads in library data, it scales the capacitance data to match the built-in element library units.

NanoTime recognizes the following unit sizes: **1ff**, **10ff**, **100ff**, and **1pf**.

Using a large design unit might cause small values to lose precision. For example, a current value of 0.23759 mA would become 237.59 uA if the current unit specified in the **lib_current_unit** variable is set to **1uA**. In this case, no information is lost. However, if the current unit is set to **1A**, the value becomes 0.000238 A, losing some digits of precision. Ensure that your design units are appropriate for your analysis needs.

You must set the parameter before you use the **link_design** command. Changing the parameter after design linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
lib_current_unit(3)
lib_resistance_unit(3)
lib_time_unit(3)
lib_voltage_unit(3)
```

lib_current_unit

Sets the unit size for electrical current.

TYPE

string

DEFAULT

1mA

DESCRIPTION

This parameter specifies the electrical current units used in the design (for example, 1A for one amp or 1mA for one milliamp). NanoTime places the specified unit size into the built-in element library when you link the design with the **link_design** command. When NanoTime reads in library data, it scales the electrical current data to match the built-in element library units.

NanoTime recognizes the following unit sizes: **1uA**, **10uA**, **100uA**, **1mA**, **10mA**, **100mA**, and **1A**.

Using a large design unit might cause small values to lose precision. For example, a current value of 0.23759 mA would become 237.59 uA if the current unit specified in the **lib_current_unit** variable is set to **1uA**. In this case, no information is lost. However, if the current unit is set to **1A**, the value becomes 0.000238 A, losing some digits of precision.

This issue is especially important if you are creating composite current source (CCS) models. You must verify that your **lib_current_unit** variable setting does not have an adverse effect on the accuracy of current values stored in the model. For best results, use a value of **1uA** for the **lib_current_unit** variable when you are creating CCS models.

You must set the parameter before you use the **link_design** command. Changing the parameter after design linking has no effect unless you execute the **link_design** command again.

SEE ALSO

link_design(2)
lib_capacitance_unit(3)
lib_resistance_unit(3)
lib_time_unit(3)
lib_voltage_unit(3)

lib_exclude_self_load_for_ccs

Specifies whether the capacitance of the output pin should be excluded from the total load driven by the pin that is used for looking up CCS tables during ETM consumption.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime excludes the self load of the output pin from the total load.

SEE ALSO

`extract_model(2)`

lib_pin_steady_state_resistance_above_high

Specifies the default steady-state holding resistance value to be used for "above high" injection noise calculation on nets driven by a model.

TYPE

float

DEFAULT

0.1

DESCRIPTION

The **update_noise** command computes injection noise on victim nets caused by coupled aggressors. A steady-state holding resistance value is used for this calculation when the victim net is driven by a model. Set this variable to the desired resistance value such that it is equal to or greater than 0.01 in ohms. The value set by this variable may be overriden using the **set_steady_state_resistance** command.

SEE ALSO

```
update_noise(2)
set_steady_state_resistance(2)
lib_pin_steady_state_resistance_above_low(3)
lib_pin_steady_state_resistance_below_low(3)
lib_pin_steady_state_resistance_below_high(3)
```

lib_pin_steady_state_resistance_above_low

Specifies the default steady-state holding resistance value to be used for "above low" injection noise calculation on nets driven by a model.

TYPE

float

DEFAULT

0.1

DESCRIPTION

The **update_noise** command computes injection noise on victim nets caused by coupled aggressors. A steady-state holding resistance value is used for this calculation when the victim net is driven by a model. Set this variable to the desired resistance value such that it is equal to or greater than 0.01 in ohms. The value set by this variable may be overriden using the **set_steady_state_resistance** command.

SEE ALSO

```
update_noise(2)
set_steady_state_resistance(2)
lib_pin_steady_state_resistance_above_high(3)
lib_pin_steady_state_resistance_below_low(3)
lib_pin_steady_state_resistance_below_high(3)
```

lib_pin_steady_state_resistance_below_high

Specifies the default steady-state holding resistance value to be used for "below high" injection noise calculation on nets driven by a model.

TYPE

float

DEFAULT

0.1

DESCRIPTION

The **update_noise** command computes injection noise on victim nets caused by coupled aggressors. A steady-state holding resistance value is used for this calculation when the victim net is driven by a model. Set this variable to the desired resistance value such that it is equal to or greater than 0.01 in ohms. The value set by this variable may be overriden using the **set_steady_state_resistance** command.

SEE ALSO

```
update_noise(2)
set_steady_state_resistance(2)
lib_pin_steady_state_resistance_above_low(3)
lib_pin_steady_state_resistance_above_high(3)
lib_pin_steady_state_resistance_below_low(3)
```

lib_pin_steady_state_resistance_below_low

Specifies the default steady-state holding resistance value to be used for "below low" injection noise calculation on nets driven by a model.

TYPE

float

DEFAULT

0.1

DESCRIPTION

The **update_noise** command computes injection noise on victim nets caused by coupled aggressors. A steady-state holding resistance value is used for this calculation when the victim net is driven by a model. Set this variable to the desired resistance value such that it is equal to or greater than 0.01 in ohms. The value set by this variable may be overriden using the **set_steady_state_resistance** command.

SEE ALSO

```
update_noise(2)
set_steady_state_resistance(2)
lib_pin_steady_state_resistance_above_low(3)
lib_pin_steady_state_resistance_above_high(3)
lib_pin_steady_state_resistance_below_high(3)
```

lib_resistance_unit

Specifies the unit size for electrical resistance.

TYPE

string

DEFAULT

1kohm

DESCRIPTION

This parameter specifies the electrical resistance units used in the design (for example, 1kohm or 1ohm). NanoTime places the specified unit size into the built-in element library when you link the design with the **link_design** command. When NanoTime reads in library data, it scales the electrical resistance data to match the built-in element library units.

NanoTime recognizes the following unit sizes: **1ohm**, **10ohm**, **100ohm**, and **1kohm**.

Using a large design unit might cause small values to lose precision. For example, a current value of 0.23759 mA would become 237.59 uA if the current unit specified in the **lib_current_unit** variable is set to **1uA**. In this case, no information is lost. However, if the current unit is set to **1A**, the value becomes 0.000238 A, losing some digits of precision. Ensure that your design units are appropriate for your analysis needs.

You must set the parameter before you use the **link_design** command. Changing the parameter after design linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
lib_current_unit(3)
lib_capacitance_unit(3)
lib_time_unit(3)
lib_voltage_unit(3)
```

lib_time_unit

Specifies the time unit size.

TYPE

string

DEFAULT

1ns

DESCRIPTION

This parameter specifies the time units used in the design (for example, 1ns for nanoseconds or 1ps for picoseconds). NanoTime places the specified unit size into the built-in element library when you link the design with the **link_design** command. When NanoTime reads in library data, it scales the time data to match the built-in element library units.

NanoTime recognizes the following unit sizes: **1ps**, **10ps**, **100ps**, and **1ns**.

You must set the parameter before you use the **link_design** command. Changing the parameter after design linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
lib_capacitance_unit(3)
lib_current_unit(3)
lib_resistance_unit(3)
lib_voltage_unit(3)
```

lib_voltage_mismatch_tolerance

Specifies the allowable difference between the rail voltage of the trigger transition and the nominal voltage of the library.

TYPE

string

DEFAULT

0.001

DESCRIPTION

NanoTime does not support scaling the delay when the library has a different nominal voltage than the rail voltage used for the trigger transition. To ensure accuracy, NanoTime computes the delay only when the voltage difference is smaller than the limit specified with this variable.

Increase the variable value if you want to accept a larger voltage difference. However, NanoTime does not perform any scaling based on the voltage difference.

SEE ALSO

```
oc_global_voltage(3)
lib_time_unit(3)
```

lib_voltage_unit

Specifies the unit size for voltage.

TYPE

string

DEFAULT

1V

DESCRIPTION

This parameter specifies the voltage units used in the design (for example, 1V for volts or 1mV for millivolts). NanoTime places the specified unit size into the built-in element library when you link the design with the **link_design** command. When NanoTime reads in library data, it scales the voltage data to match the built-in element library units.

NanoTime recognizes the following unit sizes: **1mV**, **10mV**, **100mV**, and **1V**.

Using a large design unit might cause small values to lose precision. For example, a current value of 0.23759 mA would become 237.59 uA if the current unit specified in the **lib_current_unit** variable is set to **1uA**. In this case, no information is lost. However, if the current unit is set to **1A**, the value becomes 0.000238 A, losing some digits of precision. Ensure that your design units are appropriate for your analysis needs.

You must set the parameter before you use the **link_design** command. Changing the parameter after design linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
lib_capacitance_unit(3)
lib_current_unit(3)
lib_resistance_unit(3)
lib_time_unit(3)
```

library_path

A list of directories in which to locate subcircuit references.

TYPE

string

DEFAULT

""

DESCRIPTION

The **library_path** variable specifies where and in what order to look for SPICE library files during pattern matching with the **read_pattern** command.

SEE ALSO

`link_path(3)`
`search_path(3)`
`read_pattern(2)`

link_capacitor_term1_pin_name

Specifies a list of pin names that will map to terminal 1 of a capacitor in the default library.

TYPE

string

DEFAULT

term1

DESCRIPTION

The **link_capacitor_term1_pin_name** variable specifies the pin name for terminal 1 of a capacitor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_capacitor_term2_pin_name(3)
```

link_capacitor_term2_pin_name

Specifies a list of pin names which will map to terminal 2 of a capacitor in the default library.

TYPE

string

DEFAULT

term2

DESCRIPTION

The **link_capacitor_term2_pin_name** variable specifies the pin name for terminal 2 of a capacitor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_capacitor_term1_pin_name(3)
```

link_case

Specifies the case mode to be used by the netlist reader and linker.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies the case mode to be used by the netlist reader and linker. The following settings are allowed:

"" (empty string)	case-sensitive
lower	lowercase
upper	uppercase

An empty string (the default) means that no case shifting occurs, so that linking is case-sensitive.

To shift all names to lowercase, set the variable to **lower**. To shift all names to uppercase, set the variable to **upper**. The variable must be set before the **link_design** command is used.

For example, to select lowercase linking:

```
nt_shell> set link_case lower
```

SEE ALSO

```
link_design(2)
link_model_case_sensitive(3)
```

link_control_options

Specifies netlist parsing and linking options.

TYPE

string

DEFAULT

""

DESCRIPTION

The **link_control_options** variable specifies netlist parsing and linking options. The format of the variable is a white-space delimited list of control options. Each control option must have one of the following three forms:

```
string
string:string
string:"string string ..."
```

These options control design linking, so you must set the variable before you use the **link_design** command. Supported link control options include the following:

```
discard_subckt_inst -- removes part of a design from the top design
global_override_port -- handles global and subcircuit port nodes
ignore_excess_pin -- handles excess subcircuit pins
keep_unc_node -- keeps unconnected nodes
min_cap_value -- minimum allowable cap value
min_gndcap_value -- minimum allowable grounded cap value
min_fcap_value -- minimum allowable floating cap value
set_message_limit -- limits the number of warning messages
strip_leading_x -- strips leading x from HSPICE names
vdd_gnd_prefix_node -- recognizes VDD and GND prefixes
```

For more information about these options, see the Circuit Simulation and Analysis Tools Reference Guide, available on SolvNet (http://solvnet.synopsys.com) in the NanoSim collection of manuals.

These control options are not supported, or their functionality is superseded by other NanoTime controls:

```
hier_separator -- use set hierarchy_separator
max_res_value -- internal default value
min_res_value -- internal default value
split_fcap -- use link_design -coupling_reduction_factor
top_module -- use link_design design_name
```

The **append** command is convenient for managing the **link_control_options** string. The following example shows how to append an option onto the variable.

```
nt_shell> append link_control_options \
          { min_cap_value:0.005 }
min_cap_value:0.005
```

To limit the number of warning messages produced during linking, use a command similar to the following:

```
nt_shell> append link_control_options \
          {set_message_limit:"0x2020107f 10"}
```

This example limits "0x2020107f" type warning messages to 10.

There are other NanoTime variables that also control linking options. See their man pages for more information.

SEE ALSO

```
link_design(2)
bus_notation(3)
default_top_name(3)
hierarchy_separator(3)
link_case(3)
link_prefer_model(3)
link_prefer_model_port(3)
```

link_diode_base_pin_name

Specifies a list of names which will map to the base pin of a diode.

TYPE

string

DEFAULT

ANODE

DESCRIPTION

This parameter specifies the pin name of the base pin of a diode. This name is used to back-annotate diodes using parasitic files (DSPF or DPF).

You must set this parameter before you link the design with the **link_design** command. Changing the parameter after linking has no effect unless you run the **link_design** command again.

SEE ALSO

link_design(2)
link_diode_emitter_pin_name(3)

link_diode_emitter_pin_name

Specifies a list of names which will map to the emitter pin of a diode.

TYPE

string

DEFAULT

CATHODE

DESCRIPTION

This variable specifies the pin name of the emitter pin of a diode. The name is used to back-annotate the diode using parasitic files (DSPF or DPF).

You must set this variable before you link the design with the **link_design** command. Changing the variable value after linking has no effect unless you run the **link_design** command again.

SEE ALSO

link_design(2)
link_diode_base_pin_name(3)

link_discard_subckt_contents

Specifies the names of SPICE subcircuits that are recognized as intentionally discarded subcircuits at the **read_parasitics** command.

TYPE

string

DEFAULT

""

DESCRIPTION

When you specify subcircuit names with the **link_discard_subckt_contents** variable, NanoTime ignores the contents of the specified subcircuits and replaces them with empty subcircuits that have boundary pins. In addition, all nets are preserved between the subcircuit boundary pins and the rest of the design.

This operation is different from using the **link_control_options** variable to discard SPICE subcircuits. In that case, nets between the discarded subcircuits and the rest of the design are not retained and boundary pins are not defined.

If RC elements in a parasitics netlist are attached to pins inside a discarded subcircuit, the NanoTime tool discards the affected RC elements and issues error messages about unresolved pins when the **read_parasitics** command is executed. The NanoTime run continues, but ignoring parasitic elements might affect delay accuracy.

To avoid the unresolved pin errors, use the **link_discard_subckt_contents** variable to discard subcircuits (do not use the **link_control_options** variable). Then set the **parasitics_enable_mapping_unresolved_pins** variable to **true**. In this case, NanoTime preserves the parasitics attached to the boundary pins of the discarded subcircuit or to other pins (interior to the subcircuit) that are connected to the boundary pins. The preserved parasitics are reassigned to the associated boundary pin. All other parasitics attached to pins inside the discarded subcircuit are ignored.

Coupling capacitances in the parasitic netlist between nets outside a discarded subcircuit and nets inside the subcircuit are replaced with grounded capacitances on the outside nets.

The **link_discard_subckt_contents** variable must be set before executing the **link_design** command.

SEE ALSO

```
read_parasitics(2)
parasitics_enable_mapping_unresolved_pins(3)
link_control_options(3)
```

link_enable_corner_specific_wrapper_subckts

Specifies whether corner-specific wrapper subcircuit processing is enabled during the **check_design** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, if transistor wrapper subcircuits are present, NanoTime creates corner-specific versions of each transistor wrapper subcircuit for each technology corner specified with the **set_technology** command. You can disable this feature by setting the **link_enable_corner_specific_wrapper_subckts** variable to **false**.

SEE ALSO

`set_technology(2)`

link_enable_metaencrypted_models

Enables or disables the reading of device models encrypted with the Synopsys utility called metaencrypt.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime reads in device model files that have been encrypted with the metaencrypt Synopsys utility.

This variable must be set before executing the **link_design** command.

To enable support for metaencrypted device models, you must have a NanoTime Ultra license.

SEE ALSO

`link_design(2)`

link_enable_netlist_spice_model_linking

Specifies whether the **link_design** command links transistors to the models in the input SPICE deck.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default) and if there are SPICE models present in the input SPICE deck, the **link_design** command links the transistors in the netlist to the SPICE transistor models. The **link_design** command creates a technology registry entry to hold the SPICE models and assigns the technology name specified by the **tech_netlist_spice_model_name** variable.

If the **link_enable_netlist_spice_model_linking** variable is set to **false**, the **link_design** command still reads in the SPICE models, but it does not link the transistors to the models. You can manually link the transistors to the desired models by using the **set_technology** command.

To view a list of the current technology registry entries, use the **list_technology** command.

SEE ALSO

```
link_design(2)
list_technology(2)
read_spice_model(2)
```

```
read_techfile(2)
set_technology(2)
tech_netlist_spice_model_name(3)
```

link_enable_wrapper_subckt_parasitics

Specifies whether embedded RC parasitics contained in wrapper subcircuits are preserved during the **link_design** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime preserves embedded RC parasitics found in transistor wrapper subcircuits. However, for design flows where corner-specific wrapper subcircuit processing is required, the wrapper subcircuit parasitics must be ignored during the **link_design** command. For this case, the **link_enable_wrapper_subckt_parasitics** variable must be set to **false**.

This variable cannot be changed after the **link_design** command.

The corner-specific wrapper subcircuit parasitics flow has the following restrictions:

- The **write_spice** command requires the **-insert_model** option for an accurate SPICE deck.
- If the **check_design** command must be rerun, all parasitics must be removed and then read in again.

SEE ALSO

```
link_design(2)
check_design(2)
link_enable_corner_specific_wrapper_subckts(3)
parasitics_wrapper_subckt_technology(3)
```

link_gnd_alias

Specifies the net names that are considered ground nets.

TYPE

string

DEFAULT

gnd vss 0 !gnd !vss gnd! vss! vss1 vss2 vss#

DESCRIPTION

This parameter specifies the net names that are automatically assigned to be ground nets when the design is linked with the **link_design** command. Matching is case-insensitive, and the wildcard characters '*' and '?' are supported. To make a particular net name a ground net, include the net name in the variable string before you use the **link_design** command. An alternative method to accomplish the same thing is to use the command **set_supply_net -gnd**.

The supply net change affects topology recognition. For this reason, if you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
match_topology(2)
set_supply_net(2)
link_vdd_alias(3)
```

link_grounded_capacitor_term1_pin_name

Specifies a list of pin names which will map to terminal 1 of a grounded capacitor in the default library.

TYPE

string

DEFAULT

term1

DESCRIPTION

This parameter specifies the pin name for terminal 1 of a grounded capacitor element in the default library. The first name in the list will be used in reports on pin objects (for example, when you use the **report_cell -connections -verbose** command).

You must set this parameter before you link the design with the **link_design** command. Changing the parameter after linking has no effect unless you run the **link_design** command again.

SEE ALSO

`link_design(2)`

link_inductors_as_resistors

Specifies whether the **link_design** command will convert inductors in the input netlist into parasitic resistors.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to true (the default), the **link_design** command converts inductors into parasitic resistors.

If the variable is set to false, the **link_design** command issues an error if an inductor is in the input netlist, and the command fails.

SEE ALSO

`link_design(2)`

link_model_case_sensitive

Specifies whether to use case-sensitive or case-insensitive string comparison when linking.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable specifies whether case-sensitive or case-insensitive string comparison is used when linking timing models to timing model instances.

Setting this variable to **false** selects case-insensitive linking.

SEE ALSO

`link_design(2)`
`link_case(3)`

link_nmos_alias

Specifies the names that are recognized as NMOS transistor model names.

TYPE

string

DEFAULT

nfet n* *nch*

DESCRIPTION

When the design is linked with the **link_design** command, each transistor must be recognized as an NMOS or PMOS transistor. The **link_nmos_alias** variable specifies the transistor model names that are automatically recognized as NMOS transistor names. For example, the string "n*" causes all transistor models beginning with the letter "n" to be recognized as NMOS transistors.

The alias name can also be set in the SPICE netlist using the syntax in the following example:

*nanosim tech="model_alias nch N n NCH NMOS"

EXAMPLE

The following command sets the alias names:

```
nt_shell> set link_nmos_alias "nch N n NCH NMOS"
```

SEE ALSO

link_design(2)
link_pmos_alias(3)

link_path

Specifies a list of libraries, design files, and library files used during linking.

TYPE

string

DEFAULT

*

DESCRIPTION

This variable specifies a list of libraries, design files, and library files used during linking.

When reading in a timing model, you must use the **link_path** variable to specify the path to libraries, design files, and library files during linking. The **link_design** command looks at those files and tries to resolve references in the order that the files are listed in the variable.

The **link_path** variable can contain three different types of elements: *, a library name, or a file name.

The * entry causes the **link_design** command to search all the designs loaded into NanoTime while trying to resolve references. Designs are searched in the order in which they were read. The default value of **link_path** is *.

For elements other than *, NanoTime searches for a library that has already been loaded. If that search fails, NanoTime searches for a file with the specified name in the directories specified by the **search_path** variable.

SEE ALSO

```
link_design(2)
search_path(3)
```

link_pmos_alias

Specifies the names that are recognized as PMOS transistor model names.

TYPE

string

DEFAULT

pfet p* *pch*

DESCRIPTION

When the design is linked with the **link_design** command, each transistor must be recognized as an NMOS or PMOS transistor. The **link_pmos_alias** variable specifies the transistor model names that are automatically recognized as PMOS transistor names. For example, the string "p*" causes all transistor models beginning with the letter "p" to be recognized as PMOS transistors.

The alias names can also be set in the SPICE netlist using the syntax in the following example:

*nanosim tech="model_alias pch P p PCH PMOS"

EXAMPLE

The following command sets the alias names:

```
nt_shell> set link_pmos_alias "pch P p PCH PMOS"
```

SEE ALSO

link_design(2)
link_nmos_alias(3)

link_polynomial_capacitors_as_capacitors

Specifies whether the **link_design** command converts polynomial capacitors in the input netlist into constant parasitic capacitors.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to true (the default), the **link_design** command converts polynomial capacitors into constant capacitors and treats them the same as standard parasitic capacitors in the netlist. The polynomial capacitor is evaluated at one half the voltage specified in the **oc_global_voltage** variable to determine the constant capacitance.

If the variable is set to **false**, the **link_design** command issues an error if a polynomial capacitor is in the input netlist, and the command fails.

SEE ALSO

`link_design(2)`
`oc_global_voltage(3)`

link_prefer_model

Specifies a list of subcircuit definition names for which a timing model is used in place of the subcircuit definition, with position-based port mapping.

TYPE

string

DEFAULT

""

DESCRIPTION

The **link_prefer_model** variable specifies a list of subcircuit definition names for which a timing model is preferred. For each name in the list, if a timing model exists, it is used instead of the subcircuit definition. You must set the variable before you use the **link_design** command.

The format of the variable is a white-space delimited list of names. Wildcard characters are allowed. The **append** command is convenient for managing such a string.

Be sure to use the correct variable to specify preferred timing models, depending on whether you want to specify subcircuit names or specific instances, and whether to use position-based or name-based port mapping:

Variable	Name Type	Port Mapping
link_prefer_model	subcircuit	position-based
link_prefer_model_port	subcircuit	name-based

Substitution based on subcircuit names means that for each block matching a listed name, NanoTime uses the identically named .db model in place of the netlist, if the .db model is in the search path and has been read into memory with the **read_library** command. For example, the following command directs NanoTime to use the regblk1.db timing model rather than the regblk1 netlist:

```
nt_shell> set link_prefer_model { regblk1 }
```

Port mapping between the netlist and the timing model can be either position-based or name-based. For position-based port mapping, NanoTime assumes that the ports are defined in the netlist in the same order that the ports are defined in the timing model, irrespective of naming conventions. In that case, the port names need not match.

For name-based port mapping, NanoTime matches the port names in the netlist to the port names in the timing model, irrespective of the order in which the ports are defined. In that case, the ports can be defined in any order in the netlist and the timing model.

If there is a conflict between position-based and name-based variable settings, NanoTime uses position-based port mapping.

The following command causes all available timing models found to be used instead of subcircuit definitions.

```
nt_shell> set link_prefer_model *
*
```

In the following example, the **append** command is used to add to the string value:

```
nt_shell> append link_prefer_model { inv* }
inv*
nt_shell> append link_prefer_model { latch* }
inv*  latch*
```

SEE ALSO

```
link_prefer_model_port(3)
read_library(2)
link_design(2)
```

link_prefer_model_port

Specifies a list of subcircuit definition names for which a timing model is used in place of the subcircuit definition, with name-based port mapping.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies a list of subcircuit definition names for which a timing model is preferred. For each name in the list, if a timing model exists, it is used instead of the subcircuit definition. You must set the variable before you use the **link_design** command.

The format of the variable is a white-space delimited list of names. Wildcard characters are allowed. The **append** command is convenient for managing such a string.

Be sure to use the correct variable to specify preferred timing models, depending on whether you want to specify subcircuit names or specific instances, and whether to use position-based or name-based port mapping:

Variable	Name Type	Port Mapping
link_prefer_model	subcircuit	position-based
link_prefer_model_port	subcircuit	name-based

Substitution based on subcircuit names means that for each block matching a listed name, NanoTime uses the identically named .db model in place of the netlist, if the .db model is in the search path and has been read into memory with the **read_library** command. For example, the following command directs NanoTime to use the regblk1.db timing model rather than the regblk1 netlist:

```
nt_shell> set link_prefer_model { regblk1 }
```

Port mapping between the netlist and the timing model can be either position-based or name-based. For position-based port mapping, NanoTime assumes that the ports are defined in the netlist in the same order that the ports are defined in the timing model, irrespective of naming conventions. In that case, the port names need not match.

For name-based port mapping, NanoTime matches the port names in the netlist to the port names in the timing model, irrespective of the order in which the ports are defined. In that case, the ports can be defined in any order in the netlist and the timing model.

If there is a conflict between position-based and name-based variable settings, NanoTime uses position-based port mapping.

SEE ALSO

```
link_prefer_model(3)
read_library(2)
link_design(2)
```

link_preserve_verilog_port_directions

Specifies whether to preserve the port directions given in the Verilog netlist.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default), NanoTime preserves the port directions given in the Verilog netlist. This is appropriate when the Verilog description represents the top-level netlist of the design.

When the variable is set to **false**, NanoTime does not preserve the port directions, effectively making all the ports bidirectional. This is appropriate when the Verilog description is to be instantiated in a higher-lever SPICE netlist. In this situation, NanoTime can link the design only if the lower-level ports are bidirectional.

SEE ALSO

`set_port_direction(2)`
`trace_paths(2)`

link_resistor_term1_pin_name

Specifies a list of pin names which will map to terminal 1 of a resistor in the default library.

TYPE

string

DEFAULT

term1

DESCRIPTION

The **link_resistor_term1_pin_name** variable specifies the pin name for terminal 1 of a resistor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_resistor_term2_pin_name(3)
```

link_resistor_term2_pin_name

Specifies a list of pin names which will map to terminal 2 of a resistor in the default library.

TYPE

string

DEFAULT

term2

DESCRIPTION

The **link_resistor_term2_pin_name** variable specifies the pin name for terminal 2 of a resistor element in the default library. The first pin name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_resistor_term1_pin_name(3)
```

link_switch_resistors_as_switch_subckts

Specifies whether to replace resistors in the design with subcircuit instances that can be short circuits or open circuits, depending on the resistance value.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If the **link_switch_resistors_as_switch_subckts** variable is set to **false** (the default), NanoTime treats all design resistors the same as parasitic resistors during design linking (at the **link_design** command).

If the **link_switch_resistors_as_switch_subckts** variable is set to **true**, NanoTime can replace a design resistor with a subcircuit that represents the resistor as either a short circuit or an open circuit. In other words, a design resistor can be treated as an ideal switch that is either open or closed. Parasitic resistors are not affected by this variable.

Design resistors are modified as follows:

- If the resistance value is larger than the value of the **link_switch_resistors_open_resistance** variable, the resistor is replaced with an open circuit.
- If the resistance value is smaller than the value of the **link_switch_resistors_short_resistance** variable, the resistor is replaced with a short circuit.
- If the resistance value is equal to or between the limit values, the resistor is not changed.

SEE ALSO

link_design(2)
link_switch_resistors_open_resistance(3)
link_switch_resistors_short_resistance(3)

link_switch_resistors_open_resistance

Specifies the resistance value above which NanoTime replaces a design resistor with a subcircuit that represents the resistor as an open circuit during design linking, if the **link_switch_resistors_as_switch_subckts** variable is set to **true**.

TYPE

float

DEFAULT

1.0E+6

DESCRIPTION

If the **link_switch_resistors_as_switch_subckts** variable is set to **false** (the default), design resistors are not modified.

If the **link_switch_resistors_as_switch_subckts** variable is set to **true**, NanoTime can replace a design resistor with a subcircuit that represents the resistor as either a short circuit or an open circuit. The design resistor is modified as follows:

- If the resistance value is larger than the value of the **link_switch_resistors_open_resistance** variable, the resistor is replaced with an open circuit.
- If the resistance value is smaller than the value of the **link_switch_resistors_short_resistance** variable, the resistor is replaced with a short circuit.
- If the resistance value is equal to or between the limit values, the resistor is not changed.

The unit of the variable is ohms.

SEE ALSO

```
link_design(2)
link_switch_resistors_as_switch_subckts(3)
link_switch_resistors_short_resistance(3)
```

link_switch_resistors_short_resistance

Specifies the resistance value below which NanoTime replaces a design resistor with a subcircuit that represents the resistor as a short circuit during design linking, if the **link_switch_resistors_as_switch_subckts** variable is set to **true**.

TYPE

float

DEFAULT

1.0E-6

DESCRIPTION

If the **link_switch_resistors_as_switch_subckts** variable is set to **false** (the default), design resistors are not modified.

If the **link_switch_resistors_as_switch_subckts** variable is set to **true**, NanoTime can replace a design resistor with a subcircuit that represents the resistor as either a short circuit or an open circuit. The design resistor is modified as follows:

- If the resistance value is larger than the value of the **link_switch_resistors_open_resistance** variable, the resistor is replaced with an open circuit.
- If the resistance value is smaller than the value of the **link_switch_resistors_short_resistance** variable, the resistor is replaced with a short circuit.
- If the resistance value is equal to or between the limit values, the resistor is not changed.

The unit of the variable is ohms.

SEE ALSO

```
link_design(2)
link_switch_resistors_as_switch_subckts(3)
link_switch_resistors_open_resistance(3)
```

link_transistor_bulk_pin_name

Specifies a list of names which will map to the bulk (substrate) terminal of a transistor.

TYPE

string

DEFAULT

b

DESCRIPTION

This parameter specifies the pin name for the bulk (substrate) terminal of each transistor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_transistor_drain_pin_name(3)
link_transistor_gate_pin_name(3)
link_transistor_source_pin_name(3)
```

link_transistor_drain_pin_name

Specifies a list of names which will map to the drain terminal of a transistor.

TYPE

string

DEFAULT

d

DESCRIPTION

This parameter specifies the pin name for the drain terminal of each transistor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_transistor_bulk_pin_name(3)
link_transistor_gate_pin_name(3)
link_transistor_source_pin_name(3)
```

link_transistor_gate_pin_name

Specifies a list of names which will map to the gate terminal of a transistor.

TYPE

string

DEFAULT

g

DESCRIPTION

This parameter specifies the pin name for the gate terminal of each transistor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_transistor_bulk_pin_name(3)
link_transistor_drain_pin_name(3)
link_transistor_source_pin_name(3)
```

link_transistor_source_pin_name

Specifies a list of names which will map to the source terminal of a transistor.

TYPE

string

DEFAULT

s

DESCRIPTION

This parameter specifies the pin name for the source terminal of each transistor element in the default library. The first name in the list will be used in reports on pin objects, for example, when you use the **report_cell -connections -verbose** command.

You must set this variable before you link the design with the **link_design** command. Changing the variable after linking has no effect unless you execute the **link_design** command again.

SEE ALSO

```
link_design(2)
link_transistor_bulk_pin_name(3)
link_transistor_drain_pin_name(3)
link_transistor_gate_pin_name(3)
```

link_vdd_alias

Specifies the net names that are considered power supply nets.

TYPE

string

DEFAULT

vdd vcc !vdd !vcc vdd! vcc! vdd1 vdd2 vdd#

DESCRIPTION

This parameter specifies the net names that are automatically assigned to be power supply nets when the design is linked with the **link_design** command. Matching is case-insensitive, and the wildcard characters '*' and '?' are supported. To make a particular net name a power supply net, include the net name in the variable string before you use the **link_design** command. An alternative method to accomplish the same thing is to use the command **set_supply_net**.

The supply net change affects topology recognition. For this reason, if you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
match_topology(2)
set_supply_net(2)
link_gnd_alias(3)
```

memory_enable_parallel_hspice

Enables or disables the simulation of memory circuits using multiple parallel HSPICE processes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to true to enable the use of distributed processing for HSPICE memory circuit simulation. This must be done before the **check_design** command. By default, the variable is set to false and distributed HSPICE processing is disabled.

To use distributed processing you must perform all of the following actions:

- Set the **memory_enable_parallel_hspice** variable to true.
- Define a host file.
- Set the **distributed_processing_host_file** variable to point to the host file.
- Set the **distributed_processing_number_hspice_license** variable to a value greater than 1.

Otherwise, parallel processing will not occur regardless of the value of the **memory_enable_parallel_hspice** variable.

When distributed processing for memory circuit analysis is enabled, an additional step is taken during clock path tracing to set up all the HSPICE simulations for distributed processing. The message "Performing Memory Column Pre-fill (Max or Min) Search" is displayed.

SEE ALSO

distributed_processing_host_file(3)
distributed_processing_number_hspice_license(3)
nt_tmp_dir(3)

model_apply_oc_global_voltage

Specifies whether to use the **oc_global_voltage** variable for setting the nominal voltage in the generated timing model.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime uses the highest design voltage, if available, for setting the nominal voltage value in the model. Alternatively, you can set the **model_apply_oc_global_voltage** variable to **true** to use the value of the **oc_global_voltage** variable instead.

The **model_apply_oc_global_voltage** variable does not affect the setting of the nominal voltage from consumed timing models.

SEE ALSO

`extract_model(2)`
`oc_global_voltage(3)`

model_ccs_c1_multiplier_max_fall

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_rise(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_c1_multiplier_max_rise

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_rise(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_c1_multiplier_min_fall

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_rise(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_c1_multiplier_min_rise

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_rise(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_c2_multiplier_max_fall

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_rise(3)
```

model_ccs_c2_multiplier_max_rise

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_c2_multiplier_min_fall

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_rise(3)
model_ccs_c2_multiplier_max_rise(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_c2_multiplier_min_rise

Specifies the receiver model scale factor to be applied when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the multiplier to be applied to both pin-based and arc-based CCS receiver model capacitance values.

SEE ALSO

```
extract_model(2)
model_ccs_c1_multiplier_min_rise(3)
model_ccs_c1_multiplier_min_fall(3)
model_ccs_c1_multiplier_max_rise(3)
model_ccs_c1_multiplier_max_fall(3)
model_ccs_c2_multiplier_min_fall(3)
model_ccs_c2_multiplier_max_rise(3)
model_ccs_c2_multiplier_max_fall(3)
```

model_ccs_characterization_driver_type

Specifies the type of driver to be used at the cell inputs for characterization with the **extract_model** command.

TYPE

string

DEFAULT

snps_predriver

DESCRIPTION

This variable specifies the type of driver to be used at the cell inputs for characterization with the **extract_model** command. The variable can be set to either **ramp** (for a simple ramp) or **snps_predriver** (for the standard Synopsys predriver).

The ramp is an ideal, linear ramp with the specified transition time, defined according to the **rc_slew_*_threshold_pct_*** variables. The standard Synopsys predriver represents a more realistic driving waveform.

SEE ALSO

```
extract_model(2)
model_ccs_measured_delay_tolerance(3)
model_ccs_waveform_segment_tolerance(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
```

```
rc_slew_upper_threshold_pct_rise(3)
```

model_ccs_measured_delay_tolerance

Specifies the delay tolerance allowed when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

0.02

DESCRIPTION

This variable specifies the acceptable difference between the measured delay from simulation and the delay obtained from the CCS waveform, expressed as a fraction of the measured delay from simulation. A smaller tolerance produces a larger, but more accurate model.

SEE ALSO

```
extract_model(2)
model_ccs_characterization_driver_type(3)
model_ccs_waveform_segment_tolerance(3)
```

model_ccs_waveform_segment_tolerance

Specifies the waveform segment tolerance allowed when generating composite current source (CCS) timing data with the **extract_model** command.

TYPE

float

DEFAULT

0.005

DESCRIPTION

This variable specifies the maximum allowed voltage difference between the simulation waveform and the CCS waveform, used for selecting the CCS model point. The smaller the tolerance, the more points used in waveform tables.

SEE ALSO

```
extract_model(2)
model_ccs_characterization_driver_type(3)
model_ccs_measured_delay_tolerance(3)
```

model_db_naming_compatibility

Determines the library name format used in the .db format extracted timing model.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Extracted timing models generated by NanoTime can be written in either .lib (Liberty) or .db (Synopsys database) format. The naming of the library in these two output formats can be different. The .lib format uses the string specified for the **-name** option, while the .db format uses the string for the **-name** option with the '_lib' extension. If the **model_db_naming_compatibility** variable is set to **true** (the default), this behavior is unchanged for backward compatibility. If it is set to **false**, the .db file name follows the .lib file naming format. This can improve scripting convenience when both .lib and .db format timing models are mixed in the flow.

SEE ALSO

`extract_model(2)`

model_default_input_transition_indexes

Specifies a list of default input transition values used for building the lookup tables with the **extract_model** command.

TYPE

list

DEFAULT

"0.4 0.6 0.8"

DESCRIPTION

The **model_default_input_transition_indexes** variable specifies a default list of input transition values used for building the input transition lookup tables with the **extract_model** command.

The list must contain at least three values. The values are in the time units specified in the **lib_time_unit** variable. If you do not explicitly set the **lib_time_unit** variable, it defaults to ns (nanoseconds).

You can set input transition index values explicitly with the **set_model_input_transition_indexes** command. If you do not set the index values explicitly, NanoTime uses the index values from the **model_default_input_transition_indexes** variable.

SEE ALSO

```
set_model_load_indexes(2)
set_model_input_transition_indexes(2)
extract_model(2)
```

```
lib_time_unit(3)
model_default_load_indexes(3)
```

model_default_load_indexes

Specifies a list of default load index values used to build the output load lookup tables in extracted models.

TYPE

list

DEFAULT

"0.00 0.25 0.50"

DESCRIPTION

The **model_default_load_indexes** variable specifies a default list of output capacitance values (in ff) to use for building output load lookup tables with the **extract_model** command. The list must contain at least three values.

You can set load index values explicitly with the **set_model_load_indexes** command. If you do not set the index values explicitly, NanoTime uses the index values from the **model_default_load_indexes** variable.

NanoTime ignores the load index values specified by the **set_model_load_indexes** command and the **model_default_load_indexes** variable if all of the following conditions are true:

1. The **-library_elements** option of the **extract_model** command is set to **{nldm ccs_timing}**

2. Your design contains library cells

3. A composite current source (CCS) library cell drives an output port of the design

In this case, the tool uses the load index values of the existing CCS driver model.

SEE ALSO

```
set_model_load_indexes(2)
set_model_input_transition_indexes(2)
extract_model(2)
model_default_input_transition_indexes(3)
```

model_enable_split_early_late_sigma

Causes sigma table in a delay arc to be split into separate early and late sigma tables in the extracted timing model.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime will split an early_and_late sigma table within a delay arc into separate early and late sigma tables in the extracted timing model. The early sigma table for a max delay arc will contain zero values while the late sigma table for a min delay arc will also contain zero values.

By default (when the variable is set to **false**), delay arcs will contain simga table with the sigma_type early_and_late.

SEE ALSO

`extract_model(2)`
`model_generate_pocv_lvf(3)`

model_enable_subcritical_input_path_save

Specifies whether sub-critical paths from input ports should be saved when clock uncertainty is specified but excluded from extracted model.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime can save sub-critical paths from input ports when clock uncertainty is specified but excluded from extracted model. This is to ensure that the worst case constraints are captured in the extracted timing model. This feature can be enabled only if either **-extend_inputs** or **-when** option of **extract_model** is not specified.

SEE ALSO

`extract_model(2)`

model_enable_transparent_constraint_adjustm

Specifies whether an ETM constraint value derived from a path to an internal timing check should be adjusted based on the transparency window of the boundary transparent device on the path.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, this variable enables backward compatibility with PathMill in how "infeasible" max transparent constraint paths from data inputs are modeled. A max transparent constraint path is infeasible if the setup violation at a non-boundary timing check exceeds the transparency window of the boundary transparent device. With the variable set to true, the setup constraint value placed in the ETM is adjusted based on the boundary transparency window. This can result in an optimistic setup constraint value. Note that MXTR-031 or MXTR-060 messages will accompany internal violating paths from clocks.

SEE ALSO

`extract_model(2)`

model_exclude_clock_uncertainty

Specifies whether clock uncertainty should be excluded from setup, hold, or other constraint values in the extracted timing model, while still being used during path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime includes the user-specified clock uncertainty in all aspects of path tracing and model extraction. If you set this variable to **true**, clock uncertainty is removed from all constraint values generated in the model. However, user-specified clock uncertainty is always used during path tracing, no matter how the **model_exclude_clock_uncertainty** variable is set.

This variable does not affect uncertainty from path-based slack adjustment analysis.

SEE ALSO

```
extract_model(2)
remove_clock_uncertainty(2)
set_clock_uncertainty(2)
```

model_exclude_self_load_for_ccs

Specifies whether the capacitance of the output pin should be excluded from CCS load index and max capacitance during ETM generation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime excludes the self load of the output pin from the CCS load index and max capacitance. The exclusion applies only if CCS timing models are generated. If NLDM only models are merged with NLDM plus CCS models, then the max capacitance will be the max external load.

SEE ALSO

`extract_model(2)`

model_generate_clock_arc_attributes

Specifies whether user attributes should be added to setup, hold and clock to output arcs in the timing model to indicate the clock parameters in the circuit when they are different from those in the model. Each attribute has three parts: <clock_name><sep><edge><sep><clock_port> The <sep> character is specified with the model_name_separator_char variable. The capture clock attributes are part of setup/hold constraint arcs. The launch clock attributes are part of edge triggered clock->out arcs.

TYPE

string

DEFAULT

false

DESCRIPTION

By default NanoTime does not add clock parameters as arc attributes in the timing model.

To determine the current value of this variable, type the following:

printvar model_generate_clock_arc_attributes

SEE ALSO

extract_model(2)
set_clock_uncertainty(2)
model_name_separator_char(3)

model_generate_pocv_lvf

Specifies whether POCV data is represented using the Liberty Variation Format tables in the generated timing model.

TYPE

string

DEFAULT

true

DESCRIPTION

By default, NanoTime uses the Liberty Variation Format to incorporate variation values from parametric on-chip variation (POCV) analysis into timing models, as follows:

- Nominal delay values are written into the **cell_rise** and **cell_fall** tables of the **timing()** section of the model.
- Delay variation values are written into the **rise_constraint** and **fall_constraint** tables.
- Delay variation values are written into the **ocv_sigma_cell_rise** and **ocv_sigma_cell_fall** tables.
- Delay variation values are written into the **ocv_sigma_rise_constraint** and **ocv_sigma_fall_constraint** tables.

Alternatively, you can set the **model_generate_pocv_lvf** variable to **false** to include the worst-case variation in the delay and constraint values saved in the **cell_rise**, **cell_fall**, **rise_constraint**, and **fall_constraint** tables.

SEE ALSO

set_variation_parameters(2)
extract_model(2)

model_include_all_input_signal_levels

Includes **voltage_map** and **input_signal_level** entries for all input ports in the extracted timing model.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime writes **voltage_map** and **input_signal_level** entries in the timing model for all input ports regardless of their rail voltage values.

By default (when the variable is set to **false**), these entries are not included for input ports whose rail voltage is the nominal rail voltage (in other words, equal to the value of the **oc_global_voltage** variable).

SEE ALSO

```
extract_model(2)
set_model_input_transition_indexes(2)
```

model_include_clock_to_clock_constraints

Specifies whether setup or hold constraints for clocks are included in the extracted timing model.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime excludes setup, hold, and other clock-to-clock sequential constraints from the extracted timing model.

If this variable is set to **true**, setup and hold timing checks between clocks are included in the extracted timing model as corresponding clock-to-clock constraint arcs. Included clock-to-clock constraint arcs are only created for timing checks where the checked pin is driven by a clock path. Paths that start from a clock but go through latches or registers do not give rise to such clock-to-clock constraint arcs in the timing model.

The extracted timing model from NanoTime is context-dependent on clocks. That means the clock-to-clock constraints are dependent on clock waveforms that are provided at the input. Clock context dependency applies to all delay and constraint arcs.

SEE ALSO

`extract_model(2)`

model_include_ler_for_max_clock_arcs

Specifies whether latch error recovery is included in clock-to-output delay arcs of the model. It only applies when the underlying clock-to-output paths go through transparent devices.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime includes latch error recovery in clock-to-output delay arcs.

SEE ALSO

`extract_model(2)`

model_name_separator_char

Specifies a delimiter character used to separate different fields in the names of internal pins in the model.

TYPE

string

DEFAULT

"_"

DESCRIPTION

This variable specifies a delimiter character used to separate different fields in the names of internal pins in the model. Naming conventions apply to the model latch clock and data pins. Their names contain fields such as the associated port name, the clock name, and the closing or opening edge of the clock, which are separated by the specified separator. The variable can be set to a string of multiple characters, but a single character is recommended to keep the length of the pin names manageable.

SEE ALSO

```
set_model_load_indexes(2)
set_model_input_transition_indexes(2)
extract_model(2)
model_default_input_transition_indexes(3)
model_default_load_indexes(3)
```

model_one_related_power_ground_pin

Specifies whether only one related power or ground pin can be specified for a model pin.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

NanoTime can be instructed to output only one related power pin and one related ground pin by setting this variable to true. Otherwise, it outputs all the related power and ground pins reachable from the model pin. Primary PG pins are given priority over internal PG pins (i.e., pins of virtual supplies). If there are multiple primary PG pins, then the first one encountered is chosen.

SEE ALSO

`extract_model(2)`

model_save_path_characterization_data

Specifies whether or not to save path characterization data generated by the **extract_model** command when saving a session with the **save_session** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime does not save modeling path characterization data when a session is saved. To do so, set the **model_save_path_characterization_data** variable to **true**.

SEE ALSO

`extract_model(2)`

model_slew_derate

Specifies the slew derate factor to be applied when generating a timing model with the **extract_model** command.

TYPE

float

DEFAULT

1.0

DESCRIPTION

The slew derate factor applies to all transition indexes, transition values and the max transitions for pins.

SEE ALSO

`extract_model(2)`

mos_enable_lod

Enables analysis of MOS transistor length-of-diffusion (LOD) effects.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default), NanoTime considers length-of-diffusion (LOD) effects in the transistor model when it creates a new model for a resized transistor. These effects are specified by the SA, SB, and SW parameters in the SPICE model. When this variable is set to **false**, NanoTime ignores the length-of-diffusion parameters. A setting of **true** results in higher accuracy at the cost of more runtime and memory usage.

You can specify the length-of-diffusion parameters for specified transistors in the design by using the **set_transistor_parameter** command.

SEE ALSO

`set_transistor_parameter(2)`
`mos_enable_wpe(3)`

mos_enable_wpe

Enables analysis of MOS transistor well proximity effects (WPE).

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default) NanoTime considers the well proximity effects (WPE) in the transistor model when it creates a new model for a resized transistor. These effects are specified by the SCA, SCB, SCC, and SC parameters in the SPICE model. When this variable is set to false, NanoTime ignores the WPE parameters. A setting of **true** results in higher accuracy at the cost of more runtime and memory usage.

You can specify the WPE parameters for specified transistors in the design by using the **set_transistor_parameter** command.

SEE ALSO

set_transistor_parameter(2)
mos_enable_lod(3)

nonlinear_waveform_detection_ratio

Specifies the threshold used to determine whether to use a nonlinear or linear waveform model for timing analysis.

TYPE

float

DEFAULT

5

DESCRIPTION

By default, NanoTime converts each input transition waveform to a linear waveform for simulation of a stage. However, you can optionally use the **set_nonlinear_waveform** command to instruct NanoTime to use a piecewise linear (nonlinear) waveform model instead, for better accuracy at the cost of more runtime and memory.

In certain operating modes set with the **set_nonlinear_waveform** command, NanoTime checks the smoothness of input transition waveforms to decide which type of waveform model to use for analysis. The algorithm compares the slopes of two adjacent piecewise linear segments of the waveform, A and B. It looks at the larger slope ratio, either A/B or its inverse B/A, and compares this ratio to the threshold value (5.0 by default). A ratio smaller than this threshold indicates a smooth waveform; a larger ratio indicates a nonlinear waveform.

A glitch in the waveform can cause the ratio A/B to have a negative value. In that case, the waveform is considered nonlinear.

You can also set the comparison threshold with the **-threshold** option of the **set_nonlinear_waveform** command. If the **-threshold** option is not specified in the command, the threshold is determined by the **nonlinear_waveform_detection_ratio** variable.

The threshold must be a value from 1.0 to 10000. A value of 1.0 effectively forces all waveforms to be

considered nonlinear. A large value like 1000 causes most waveforms to be considered smooth.

SEE ALSO

set_nonlinear_waveform(2)
nonlinear_waveform_samples(3)

nonlinear_waveform_samples

Specifies the number of data points used to represent each nonlinear transition waveform.

TYPE

float

DEFAULT

10

DESCRIPTION

NanoTime converts each input transition waveform to a linear waveform for simulation of a stage. For higher accuracy, you can instruct NanoTime to save nonlinear transition waveforms by using the **set_nonlinear_waveform** command.

The **nonlinear_waveform_samples** variable specifies the number of data points used to represent each nonlinear transition waveform. It must be an integer from 5 to 100. A larger number results in a more accurate model, at the cost of more memory and runtime.

You can override the number of samples used for specified nets by using the **-samples** option of the **set_nonlinear_waveform** command.

SEE ALSO

`set_nonlinear_waveform(2)`
`nonlinear_waveform_detection_ratio(3)`

nt_tmp_dir

Specifies the directory for storing temporary files.

TYPE

string

DEFAULT

. (period character)

DESCRIPTION

This variable specifies the directory that NanoTime uses to store temporary files. By default, the value is "." (the current directory). You can set this variable to any directory, such as /tmp.

SEE ALSO

oc_global_voltage

Specifies the default power supply voltage.

TYPE

float

DEFAULT

-1.0

DESCRIPTION

The **oc_global_voltage** variable specifies the default power supply voltage for the design netlist. Set this variable early in the design flow, for example, before the **set_technology** or **check_topology** commands.

If you leave **oc_global_voltage** at its default setting of -1.0, the **check_design** command automatically sets it to the correct voltage if your design uses only one rail voltage. The **check_design** command reports an error if it cannot determine the correct voltage to use. The **check_design** command issues an error if the value is set to zero or is negative. A warning is issued if the current setting is outside of the range of rail voltages in the design.

The **set_input_transition** command uses the **oc_global_voltage** value as the rail voltage for generating input waveforms unless you include the **-rail_voltage** option with the **set_input_transition** command.

When creating extracted timing models, NanoTime uses the highest design voltage for setting the nominal voltage in the model. To use the value of the **oc_global_voltage** variable instead, set the **model_apply_oc_global_voltage** variable to **true**.

To view a list of the current technology registry entries and their power supply voltages, use the **list_technology** command.

SEE ALSO

```
check_design(2)
link_design(2)
list_technology(2)
set_input_transition(2)
tech_default_voltage(3)
model_apply_oc_global_voltage(3)
```

parasitics_accept_node_name_net_name

Enables recognition of RC subnodes which do not have subnode numbers to support the reading of parasitic files with such RC subnodes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

In DSPF and SPEF parasitic files, RC subnodes have subnode numbers. However, some extraction tools create parasitic files that have RC subnodes without subnode numbers when some options are applied. For example, StarRC creates such files with the option "NETLIST_NODENAME_NETNAME:YES."

When NanoTime reads in such parasitic files, it produces warning or error messages like PARA-006, PARA-040, and DES-002.

The parasitics_accept_node_name_net_name variable must be set to **true** to avoid these messages and annotate parasitic data correctly.

SEE ALSO

`read_parasitics(2)`

parasitics_allow_mos_gate_delta_resistance

Specifies if negative parasitic resistances that may arise from the modeling of transistor gate pins are allowed to be read and used.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

The modeling of transistor gate pins may cause negative parasitic resistances to exist when the StarRC command MOS_GATE_DELTA_RESISTANCE is selected during parasitic extraction. See the StarRC documentation for additional details. This variable is true by default, and this variable must be true for NanoTime to read and use negative parasitic resistances that may arise from the modeling of transistor gate pins. If this variable is false, the negative parasitic resistances are effectively shorted in NanoTime.

parasitics_allow_spf_net_override

Specifies whether the **set_load** command is allowed to override back-annotated parasitic data.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **read_parasitics** command back-annotates the design with parasitic capacitance data from an SPEF or DSPF file. By default, a net that has been back-annotated with capacitance data from any standard parasitic file cannot be changed by the **set_load** command. Attempting to do so generates a warning message.

To allow the **set_load** command to override back-annotated parasitic data, set the **parasitics_allow_spf_net_override** variable to **true**. In that case, using the **set_load** command overrides any back-annotated data without issuing a warning.

SEE ALSO

`read_parasitics(2)`
`set_load(2)`

parasitics_cap_warning_threshold

Specifies a capacitance threshold beyond which a warning message is issued during the reading of a parasitics file.

TYPE

float

DEFAULT

0.0

DESCRIPTION

If this variable is set with a value greater than 0.0, the **read_parasitics** command issues a PARA-014 warning if it finds in the parasitics file a capacitance value, in picofarads, greater than this threshold. The default is 0.0, in which case no checking is done.

Use this variable to detect large, unexpected capacitance values written to parasitics files by other applications. The capacitor is still used, but you can quickly find it in the parasitics file.

To define an analogous resistance threshold, set the **parasitics_res_warning_threshold** variable.

SEE ALSO

```
read_parasitics(2)
parasitics_res_warning_threshold(3)
PARA-014(n)
```

parasitics_completion_capacitance

Specifies the capacitance values used for completing partially annotated nets when the command **read_parasitics -complete_with rc** is used.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **read_parasitics** command, when used with the **-complete_with** option, back-annotates the design with parasitic data and automatically completes any partially annotated nets at hierarchical pins by inserting capacitors and resistors. When the **-complete_with** type is set to **rc**, the values of the capacitors used for completion are determined by the **parasitics_completion_capacitance** variable. Set the variable to the desired capacitance value, in picofarads.

If you set the variable to zero (the default), NanoTime uses a very small value close to zero, rather than exactly zero, because the delay calculation algorithm requires nonzero element values.

SEE ALSO

`read_parasitics(2)`
`parasitics_completion_resistance(3)`

parasitics_completion_r_count_per_net_warnin

Specifies a threshold for the total count of resistor elements per net during parasitics completion beyond which a warning message is issued.

TYPE

integer

DEFAULT

50000

DESCRIPTION

During parasitics completion, NanoTime issues a **PARS-025** warning message if the total count of resistor elements for a single net exceeds this threshold.

SEE ALSO

```
complete_net_parasitics(2)
read_parasitics(2)
check_design(2)
PARS-025(n)
```

parasitics_completion_resistance

Specifies the resistance values used for completing partially annotated nets when the command **read_parasitics -complete_with rc** is used.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **read_parasitics** command, when used with the **-complete_with** option, back-annotates the design with parasitic data and automatically completes any partially annotated nets at hierarchical pins by inserting capacitors and resistors. When the **-complete_with** type is set to **rc**, the values of the resistors used for completion are determined by the **parasitics_completion_resistance** variable. Set the variable to the desired resistance value, in kohms.

If you set the variable to zero (the default), NanoTime uses a very small value close to zero, rather than exactly zero, because the delay calculation algorithm requires nonzero element values.

SEE ALSO

`read_parasitics(2)`
`parasitics_completion_capacitance(3)`

parasitics_coupling_cap_variation_max

Specifies maximum coupling capacitance variation globally.

TYPE

float

DEFAULT

0.0

DESCRIPTION

SI Delay and Noise analysis may incorporate coupling capacitance variation. Set this variable to specify maximum variation globally. For example, 0.1 corresponds to a 10% increase in coupling capacitance. This variable must be greater than or equal to 0.0.

SEE ALSO

parasitics_read_variation(3)
parasitics_coupling_cap_variation_min(3)

parasitics_coupling_cap_variation_min

Specifies minimum coupling capacitance variation globally.

TYPE

float

DEFAULT

0.0

DESCRIPTION

SI Delay and Noise analysis may incorporate coupling capacitance variation. Set this variable to specify minimum variation globally. For example, 0.1 corresponds to a 10% reduction in coupling capacitance. This variable must be greater than or equal to 0.0 and less than 1.0.

SEE ALSO

parasitics_read_variation(3)
parasitics_coupling_cap_variation_max(3)

parasitics_device_name_mapping_method

Defines the priority order of methods for mapping device names in parasitic files to the logical netlist.

TYPE

string

DEFAULT

"direct prepend_x prepend_m prepend_d prepend_r prepend_c replace_leading_m_with_x replace_leading_d_with_x replace_leading_r_with_x replace_leading_c_with_x"

DESCRIPTION

In some parasitic files, the names of devices do not match directly with the name in the logical netlist. For example, there may be a device MN1 in the parasitic file that corresponds to device XMN1 in the logical netlist.

NanoTime provides several methods for mapping these names, the priority of which is controlled by the parasitics_device_name_mapping_method variable.

This variable provides a list of methods that will be tried in order until a logical device is found that corresponds to the physical device name.

direct No mapping is done. The parasitic name is compared to the logical name exactly. Applies to all device types.

prepend_x Prepends an 'X' or 'x' to the parasitic name. (MN1 -> XMN1) Applies to all device types.

prepend_m Prepends an 'M' or 'm' to the parasitic name. (MN1 -> MMN1) Applies to MOS devices and macromodels only.

prepend_d Prepends an 'D' or 'd' to the parasitic name. (D1 -> DD1) Applies to diode devices and macromodels only.

prepend_c Prepends an 'C' or 'r' to the parasitic name. (C1 -> CC1) Applies to capacitor devices and macromodels only.

prepend_r Prepends an 'R' or 'r' to the parasitic name. (R1 -> RR1) Applies to resistor devices and macromodels only.

replace_leading_m_with_x Replaces any leading 'M' or 'm' with an 'X' or 'x'. (MN1 -> XN1) Applies to MOS devices and macromodels only.

replace_leading_d_with_x Replaces any leading 'D' or 'd' with an 'X' or 'x'. (DD1 -> XD1) Applies to diode devices and macromodels only.

replace_leading_c_with_x Replaces any leading 'C' or 'c' with an 'X' or 'x'. (CC1 -> XC1) Applies to capacitor devices and macromodels only.

replace_leading_r_with_x Replaces any leading 'R' or 'r' with an 'X' or 'x'. (RR1 -> XR1) Applies to resistor devices and macromodels only.

The default ordering has proven to be effective on the vast majority of designs, but the optimal ordering depends on the specific LVS and parasitic extraction configuration being used.

SEE ALSO

`read_parasitics(2)`

parasitics_enable_annotation_to_embedded_rc

Enables parasitic back-annotation to the pins of resistors and capacitors in a netlist.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

In the normal NanoTime parasitic back-annotation flow, all pins in the parasitic files are assumed to be connected to transistors or hierarchical subcircuits that are not resistors or capacitors. However, in some design verification flows, users put special resistor or capacitor subcircuits in the pre-layout design, then use the **read_parasitics** command to back-annotate parasitics that contain the pins of those embedded resistor or capacitor subcircuits.

If you set the **parasitics_enable_annotation_to_embedded_rc** variable to **true**, NanoTime recognizes the pins of the resistor or capacitor subcircuits in the netlist during parasitic back-annotation when you use the **read_parasitics** command.

Set this variable to **true** only when necessary because this setting can increase the memory footprint and CPU time.

SEE ALSO

`read_parasitics(2)`

parasitics_enable_drain_source_swap

Enables recognition of swapped drain and source terminals, to support the reading of fingered devices in DSPF files.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Fingered devices exist when drawn transistors in the netlist have been split into multiple transistors with the same or equivalent drive strengths as the original transistor. The splitting of a transistor into multiple parallel transistors is done after layout. When this happens, the logical pin of the drawn transistor no longer exists and is represented instead by multiple physical pins. This representation appears in the DSPF file.

In SPICE syntax, the drain, gate, and source pins must be defined for every transistor. However, because transistor source and drain terminals are interchangeable, these transistor terminals might be swapped during post-layout splitting. The original net connected to the drain pin might be connected to the source pin instead, and vice versa. To enable recognition of swapped drain/source terminals, set the **parasitics_enable_drain_source_swap** variable to **true**.

In the DSPF file, new transistors created by splitting are named by adding special characters to the existing names. To allow recognition of fingered devices, specify the special characters in the **parasitics_fingered_device_chars** variable.

SEE ALSO

```
read_parasitics(2)
parasitics_fingered_device_chars(3)
```

parasitics_enable_mapping_unresolved_pins

Enables the recognition of unresolved pins inside discarded SPICE subcircuits or timing models during parasitic back-annotation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the **read_parasitics** command is executed, parasitic RC elements assigned to pins that do not exist in the pre-layout netlist (unresolved pins) are discarded. NanoTime issues PARA-044 and DES-002 error messages for such RC elements and cannot retain detailed RC parasitics for the nets that contain these parasitics. The NanoTime run continues, but ignoring parasitic elements might affect delay accuracy.

Unresolved pins occur inside timing models or intentionally discarded subcircuits, if parasitics have been extracted for them.

To avoid the unresolved pin errors, set the **parasitics_enable_mapping_unresolved_pins** variable to **true**. In this case, NanoTime preserves the parasitics attached to boundary pins of the model or discarded subcircuit. In addition, the tool preserves parasitics attached to other pins (interior to the model or subcircuit) that are connected to the boundary pins. The preserved parasitics are reassigned to the associated boundary pin. For timing models, the original pin capacitance is also retained.

All other parasitics associated with pins inside the model or discarded subcircuit are ignored.

Coupling capacitances in the parasitic netlist between nets outside a timing model or discarded subcircuit and nets inside the model or subcircuit are replaced with grounded capacitances on the outside nets.

The **parasitics_enable_mapping_unresolved_pins** variable applies only to subcircuits discarded with

the **link_discard_subckt_contents** variable. It does not apply to subcircuits discarded using the **link_control_options** variable.

The **parasitics_enable_mapping_unresolved_pins** variable must be set before executing the **read_parasitics** command. Only DSPF parasitics netlists can be used; SPEF parasitics netlists are not supported.

SEE ALSO

```
read_parasitics(2)
link_discard_subckt_contents(3)
link_control_options(3)
link_prefer_model(3)
link_prefer_model_port(3)
```

parasitics_enable_rail_net_resistance

Enables the back-annotation of parasitic resistances on rail nets and uses them in path delay estimation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime ignores back-annotated resistances and capacitances on rail nets. However, at technology nodes of 65 nm and below, the contact resistances on rail nets can have a 5 to 15 percent impact on path delay.

NanoTime provides two methods to include the parasitics on rail nets in path delay estimation:

(1) When the **parasitics_enable_rail_contact_resistance** variable is set to **true**, NanoTime calculates the single effective contact resistance value for each pin connected to the rail nets. This flow has limitations in handling of complex resistance topologies that arise from trench contacts used in modern FinFET technologies.

(2) When the **parasitics_enable_rail_net_resistance** variable is set to **true**, NanoTime includes the complete set of back-annotated rail net resistances in path delay estimation. Complex resistance topologies including meshes, ladders, or loops are accurately modeled. Resistances on virtual rail nets are supported as well. For the correct back-annotation of resistances on virtual rail nets, make sure that power switch transistors are properly marked or recognized.

If you set more than one of these variables to **true**, the tool issues a CMD-013 error.

Even when the **parasitics_enable_rail_net_resistance** variable is enabled, capacitances on rail nets are not included in path delay estimation. Also, NanoTime does not perform an IR-drop analysis. Rail

resistances are incorporated only for the purpose of the single-stage delay calculation.

The parasitics on rail nets in SPICE decks created by the **write_spice** command might be different from the original parasitic files due to internal partitioning and reduction. The reduction tolerance for non-rail nets is controlled by the **rc_reduction_max_net_delta_delay** and **rc_reduction_min_net_delta_delay** variables. These variables do not apply to rail nets.

SEE ALSO

```
read_parasitics(2)
set_supply_net(2)
mark_power_switch(2)
check_topology(2)
parasitics_enable_rail_contact_resistance(3)
rc_reduction_min_net_delta_delay(3)
rc_reduction_max_net_delta_delay(3)
```

parasitics_fingered_device_chars

Specifies the characters used in the DSPF file to designate fingered transistors that have been split into multiple devices.

TYPE

string

DEFAULT

@ #

DESCRIPTION

Fingered devices exist when drawn transistors in the netlist have been split into multiple transistors with the same or equivalent drive strengths as the original transistor. The splitting of a transistor into multiple parallel transistors is done after layout. When this happens, the logical pin of the drawn transistor no longer exists and is represented instead by multiple physical pins. This representation appears in the DSPF file.

In the DSPF file, new transistors created by splitting are named by adding special characters to the existing names. For example an original transistor X1/X1/MP0 might be split into two transistors, X1/X1/MP0 and X1/X1/MP0@1, where the special character "@" represents an additional parallel transistor.

To allow recognition of fingered devices, specify the special characters in the **parasitics_fingered_device_chars** variable. In addition, you must enable drain/source swapping by setting the **parasitics_enable_drain_source_swap** variable to **true**.

SEE ALSO

```
read_parasitics(2)
parasitics_enable_drain_source_swap(3)
```

parasitics_ground_incomplete_coupling_cap

Recognizes incomplete coupling capacitance in a DSPF file and converts the coupling capacitance to a ground capacitance on the defined net.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

In a DSPF parasitics file, coupling capacitance is defined between two nets. In Example 1, the line "CC1 A B 0.002" is the coupling capacitance between net A and net B. The parasitics of both nets A and B also appear in the file.

```
Example 1:

...
*|NET A 1.0E-04PF
*|I (XI1/MN:d XI1/MN d X 0.0E+00PF 0 0)
*|I (XI1/MP:d XI1/MP d X 0.0E+00PF 0 0)
*|I (XI2/MN:g XI2/MN g X 0.0E+00PF 0 0)
*|I (XI2/MP:g XI2/MP g X 0.0E+00PF 0 0)
CC1 A         B    0.002

*|NET B 1.0E-04PF
*|I (XI2/MN:d XI2/MN d X 0.0E+00PF 0 0)
*|I (XI2/MP:d XI2/MP d X 0.0E+00PF 0 0)
*|I (XI3/MN:g XI3/MN g X 0.0E+00PF 0 0)
*|I (XI3/MP:g XI3/MP g X 0.0E+00PF 0 0)
...
```

When a DSPF file contains a coupling capacitor between two nets but includes the definition of only one of the two nets, the coupling capacitance is incomplete. In Example 2, "CC1 A B 0.002" is an incomplete coupling capacitance because net B does not appear in the file.

```
Example 2:

...
*|NET A 1.0E-04PF
*|I (XI1/MN:d XI1/MN d X 0.0E+00PF 0 0)
*|I (XI1/MP:d XI1/MP d X 0.0E+00PF 0 0)
*|I (XI2/MN:g XI2/MN g X 0.0E+00PF 0 0)
*|I (XI2/MP:g XI2/MP g X 0.0E+00PF 0 0)
CC1 A         B    0.002
...
```

By default, NanoTime ignores such a coupling capacitance and issues a **PARA-040** message.

To enable the recognition of an incomplete coupling capacitance, set the **parasitics_ground_incomplete_coupling_cap** variable to **true**. NanoTime recognizes the incomplete coupling capacitance and converts it to a ground capacitance on the defined net (net A in this example).

SEE ALSO

read_parasitics(2)

parasitics_min_capacitance

Specifies the minimum capacitance value (in picofarads) used for conversion of capacitance values which are smaller than the specified value during **read_parasitics**.

TYPE

float

DEFAULT

1.0e-6

DESCRIPTION

The **parasitics_min_capacitance** variable specifies the minimum capacitance value (in picofarads) that NanoTime uses in delay analysis. When a capacitor has a smaller value than the minimum, the value is converted to the minimum. This minimum value is also used in parasitic completion when the **read_parasitics** command is used with the **-complete_with** or **complete_net_parasitics** options.

SEE ALSO

`read_parasitics(2)`
`parasitics_completion_capacitance(3)`

parasitics_port_delimiter

Specifies a delimiter character used to separate different fields in the names of pins expanded from the single port name in DSPF parasitics files.

TYPE

string

DEFAULT

"_"

DESCRIPTION

Extraction tools can create parasitic files that have multiple pins with a single port. For example, if the **SHORT_PINS:NO** command is used in the StarRC command file, the StarRC parasitic extraction tool might create a file named sample.dspf with content as follows:

```
sample.dspf
*|NET vdd 0PF
*|P (vdd_1 X 0 17 50)
*|P (vdd_2 X 0 31 79)
...
R1 vdd_1 vdd_2 0.01
R2 vdd_1 vdd 0.01
...
```

To read this DSPF file in NanoTime, set the **parasitics_port_delimiter** variable as follows before executing the **read_parasitics** command:

```
set parasitics_port_delimiter "_"
```

In sample.dspf, port **vdd** has two pin definition,, pin **vdd_1** and pin **vdd_2**. When parsing the line **R1 vdd_1 vdd_2 0.01**, NanoTime matches pin **vdd_1** and pin **vdd_2** to the original port **vdd** by using the **parasitics_port_delimiter** variable setting.

The delimiter is a single character. Setting the **parasitics_port_delimiter** variable to "" or " " disables the attempt to match the corresponding port. The naming convention of the pin name must be {port name}{delimiter}{integer}.

When NanoTime reads in the parasitics file, the tool annotates parasitics on pins to the corresponding port if name matching is successful, electrically merging multiple pins.

SEE ALSO

`read_parasitics(2)`

parasitics_promote_black_box_nodes

parasitics_promote_black_box_nodes

Promotes layout-only nodes inside black-boxed subcircuits to be promoted to numbered subnodes to enable parasitics back-annotation to those nodes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Typically, connection to a subcircuit is made through named ports of the subcircuit. However, in some cases a net inside a subcircuit connects physically to a net outside the subcircuit, even though the subcircuit net is not a port. For example, a feedthrough net might pass through a subcircuit. Also, the layout versus schematic checking operation might identify layout-only nets, which do not have a corresponding net in the schematic and therefor pose problems for back-annotation of parasitics.

By default, the NanoTime tool does not allow parasitics back-annotation to internal subcircuit nodes. The tool issues a PARA-044 warning and the wire delay is zero for the nets that connect to the internal subcircuit nodes.

You can make internal subcircuit nodes available for parasitics back-annotation by setting the **parasitics_promote_black_box_nodes** variable to **true**. In this case, NanoTime calculates the wire delay for the nets that connect to the internal subcircuit nodes and no warning messages are issued for these nets. Only parasitics files in DSPF format are supported for this feature.

For layout-only nets, ensure that the **parasitics_xref_layout_instance_prefix** and **parasitics_xref_layout_net_prefix** are set to match the output of the extraction tool. The defaults of these variables correspond to the settings of the StarRC extraction tool.

RESTRICTION

This feature works only for parasitics read in DSPF.

SEE ALSO

read_parasitics(2)
link_prefer_model(3)
parasitics_xref_layout_instance_prefix(3)
parasitics_xref_layout_net_prefix(3)
parasitics_enable_mapping_unresolved_pins(3)
parasitics_suppress_message_for_objects_in_black_box(3)

parasitics_rc_count_per_net_warning_threshol

Specifies a threshold for the total count of resistor and grounded capacitor elements per net after rc reduction beyond which a warning message is issued.

TYPE

integer

DEFAULT

5000

DESCRIPTION

After rc reduction, NanoTime issues a **PARS-022** warning message if the total count of resistor and grounded capacitor elements for a single net exceeds this threshold. A **PARS-024** message may also be issued during reduction.

SEE ALSO

```
read_parasitics(2)
PARS-022(n)
PARS-024(n)
```

parasitics_read_variation

Specifies if coupling capacitance variation is to be read from a SPEF parasitics file.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

SI delay and noise analysis might incorporate coupling capacitance variation. Set this variable to true to enable coupling capacitance variation to be read from an SPEF parasitics file.

SEE ALSO

`parasitics_coupling_cap_variation_min(3)`
`parasitics_coupling_cap_variation_max(3)`

parasitics_reduction_forced_min_delta_delay

Minimum error tolerance to be used during forced RC reduction.

TYPE

float

DEFAULT

0.000001

DESCRIPTION

Minimum error tolerance to be used during forced RC reduction associated with a **PARS-024** message. Set to 0.0 to prevent any forced RC reduction from happening.

SEE ALSO

PARS-024(n)

parasitics_reduction_no_touch

Specifies whether to completely disable RC reduction.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Even with an error tolerance set to 0.0, some RC reduction may still happen due to equivalence transformations. Setting this variable to **true** will completely disable RC reduction, and the original parasitics will not be touched.

SEE ALSO

`rc_reduction_min_net_delta_delay(3)`
`rc_reduction_max_net_delta_delay(3)`

parasitics_rejection_net_size

Defines a threshold number of nodes in an annotated parasitic network beyond which the detailed network is automatically replaced by a lumped capacitance.

TYPE

int

DEFAULT

100000

DESCRIPTION

Defines a threshold number of nodes in an annotated parasitic network beyond which the detailed network is automatically replaced by a lumped capacitance. The default is *100000*.

This variable is one of a pair of variables, **parasitics_warning_net_size** and **parasitics_rejection_net_size**, that help you prevent unacceptable or unexpected run times caused by large numbers of nodes in an annotated parasitic network. If the **read_parasitics** command finds a number of nodes that exceeds the value of the **parasitics_warning_net_size** variable (default *10000*), you receive a *PARA-003* message warning you that extended run time could occur. If the **read_parasitics** command finds a number of nodes that exceeds the value of the **parasitics_rejection_net_size** variable (default *100000*), the network is rejected and automatically replaced by a lumped capacitance, and you receive a *PARA-004* message warning.

The value of the **parasitics_warning_net_size** variable is ignored if it is greater than or equal to the value of the **parasitics_rejection_net_size** variable.

To determine the current value of this variable, enter the following command:

```
nt_shell> printvar parasitics_rejection_net_size
```

SEE ALSO

```
read_parasitics(2)
parasitics_warning_net_size(3)
```

parasitics_res_warning_threshold

Specifies a resistance threshold beyond which a warning message is issued during the reading of a parasitics file.

TYPE

float

DEFAULT

0.0

DESCRIPTION

When this variable is set with a value greater than 0.0, the **read_parasitics** command issues a PARA-014 warning if it finds in the parasitics file a resistance value, in ohms, greater than this threshold. The default is 0.0, in which case no checking is done.

Use this variable to detect large, unexpected resistance values written to parasitics files by other applications. The resistor is still used, but you can quickly find it in the parasitics file.

To define an analogous capacitance threshold, set the **parasitics_cap_warning_threshold** variable.

SEE ALSO

```
read_parasitics(2)
parasitics_cap_warning_threshold(3)
PARA-014(n)
```

parasitics_suppress_dpf_inheritance

Specifies netlist transistor parameter names which are to be suppressed in macro-model device parameter annotation.

TYPE

string

DEFAULT

""

DESCRIPTION

In device parameter annotation, missing parameters in device parameter files (DPF or DSPF) are inherited from corresponding netlist transistors. Some parameters, however, should not be inherited from netlist transistor parameters for the correct interpretation of the device parameter files. For example, multiplication factor (m) should not be inherited from a netlist transistor if the extracted device parameter file (DPF or DSPF) has fingered devices.

The format for this value is a space-separated list of names.

This variable only applies to macro-model device parameter annotation.

SEE ALSO

`read_parasitics(2)`
`read_device_parameters(2)`

parasitics_suppress_message_for_objects_in_b

Enable messages regarding nets or pins described in parasitic file but do not exist in top-level netlist database because they are inside of black-boxes (timing models or discarded subcircuits).

TYPE

Boolean

DEFAULT

true

DESCRIPTION

The top-level netlist database of NanoTime does not maintain the information of nets or pins inside of black-boxes. However, parasitics can be extracted based on a transistor-level netlist, which results in all nets or pins internal to black-boxes appear in parasitic files.

By default, NanoTime suppresses messages regarding such nets or pins during the **read_parasitics** command. Set this variable to false to enable the messages.

SEE ALSO

```
read_parasitics(2)
parasitics_enable_mapping_unresolved_pins(3)
link_prefer_model(3)
link_prefer_model_port(3)
link_discard_subckt_contents(3)
```

parasitics_warning_net_size

Specifies the number of annotated nodes in a network that triggers a warning messages about long runtimes.

TYPE

integer

DEFAULT

10000

DESCRIPTION

When you use the **read_parasitics** command, if the number of annotated nodes in the design exceeds the specified number, NanoTime issues a warning message saying that the runtime could be long.

SEE ALSO

`read_parasitics(2)`

parasitics_wrapper_subckt_technology

Specifies which corner technology file is used when evaluating embedded RC parasitics contained in wrapper subcircuits.

TYPE

String

DEFAULT

""

DESCRIPTION

NanoTime can evaluate RC parasitics embedded in wrapper subcircuits during link_design or check_design. If **link_enable_wrapper_subckt_parasitics** is true, then this occurs during link_design. Otherwise, it occurs during check_design, using device parameters that may have been back-annotated from SPF or DPF.

This variable has no effect if **link_enable_wrapper_subckt_parasitics** is true, or if only one corner technology file has been defined.

If **link_enable_wrapper_subckt_parasitics** is false, and multiple corner technology files have been defined using the **set_technology** command, then this variable selects which corner tech file is used to embed the wrapper subckt parasitics.

SEE ALSO

link_design(2)

```
check_design(2)
set_technology(2)
read_spice_model(2)
link_enable_wrapper_subckt_parasitics(3)
```

parasitics_xref_layout_instance_prefix

Sets a prefix for layout device instance names that were not cross-referenced to a schematic device.

TYPE

string

DEFAULT

""

DESCRIPTION

In some parasitic files, there exist instances which are not cross-referenced to a schematic instance. Such instances are created by an extraction tool with an instance name prefix. For example, the **XREF_LAYOUT_INST_PREFIX** command in the Synopsys StarRC parasitic extraction tool sets a prefix for layout device instance names that were not cross-referenced to a schematic instance by the layout versus schematic tool.

To allow recognition of such instances, specify a prefix in the **parasitics_xref_layout_instance_prefix** variable. Ensure that this prefix matches the prefix specified by the extraction tool.

EXAMPLES

The StarRC tool **XREF_LAYOUT_INST_PREFIX** command defaults to ld_. If the default is not changed, a detailed standard parasitic format (DSPF) file might include lines such as the following:

```
Xld_XP1 Xld_XP2:S Xld_XP1:G VDD pdev_a L=0.04U W=0.32U AD= ...
Xld_XN1 Xld_XN2:S Xld_XN1:G VSS ndev_a L=0.04U W=0.20U AD= ...
```

To successfully read this file with the **read_parasitics** command, first set the **parasitics_xref_layout_instance_prefix** variable to match, as follows:

```
nt_shell> set parasitics_xref_layout_instance_prefix ld_
```

SEE ALSO

read_parasitics(2)
parasitics_xref_layout_net_prefix(3)

parasitics_xref_layout_net_prefix

Sets a prefix for parasitic net names that were not cross-referenced to a schematic net.

TYPE

string

DEFAULT

ln_

DESCRIPTION

In some parasitic files, there exist nets which are not cross-referenced to a schematic net. Such nets are created by an extraction tool with net name prefix. For example, the **XREF_LAYOUT_NET_PREFIX** command in the Synopsys StarRC parasitic extraction tool sets a prefix for layout net names that were not cross-referenced to a schematic net by the layout versus schematic tool.

To allow recognition of such nets, specify a prefix in the **parasitics_xref_layout_net_prefix** variable. Ensure that this prefix matches the prefix specified by the extraction tool.

EXAMPLES

The StarRC tool **XREF_LAYOUT_NET_PREFIX** command defaults to ln_. If the default is not changed, a detailed standard parasitic format (DSPF) file might include lines such as the following:

```
*|NET ln_002 10FF
*|I (Xld_XP1:G Xld_XP1 G|5.96e-18 0.964 15.576)
Cg1 Xld_XP1:G 0 2.25309e-16
```

```
*|NET ln_003 10FF
*|I (Xld_XN1:G Xld_XN1 G|5.96e-18 0.964 15.576)
Cg1 Xld_XN1:G 0 2.25309e-16
```

To successfully read this file with the **read_parasitics** command, leave the **parasitics_xref_layout_instance_prefix** variable set to its default, which matches the StarRC default. If needed, you would set the variable as follows:

```
nt_shell> set parasitics_xref_layout_net_prefix ln_
```

SEE ALSO

```
read_parasitics(2)
parasitics_xref_layout_instance_prefix(3)
```

pattern_merge_parallel_transistors

Specifies whether to merge parallel transistors for pattern recognition.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Pattern matching in NanoTime helps you to find and mark structures in the design. A pattern is a subcircuit usually described in a SPICE or Verilog file.

NanoTime performs pattern recognition during the **read_pattern** command. In addition, you can automate the process using script files that are executed during the **match_topology** command. You enable the use of scripts by using one of the **topo_auto_recognize_*** variables. See the variable man pages for more information.

During pattern recognition, NanoTime merges parallel transistors if the **pattern_merge_parallel_transistors** variable is set to **true** (the default). Two or more transistors are parallel if they are the same type of transistor (NMOS or PMOS) and have the same source, gate, and drain pin connections.

The **pattern_merge_parallel_transistors** variable affects the merging done for pattern recognition only. Use the **timing_merge_parallel_transistors** variable to affect the merging done for path tracing.

SEE ALSO

match_topology(2)
read_pattern(2)
timing_merge_parallel_transistors(3)
topo_auto_recognize_pattern_command_file(3)
topo_auto_recognize_user_pattern_command_file(3)
topo_auto_recognize_clock_dependent_pattern_command_file(3)
topo_auto_recognize_user_clock_dependent_pattern_command_file(3)

pbsa_KALmax

Specifies the KALmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KALmax** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment A of each timing check to obtain the adjustment value for that segment, for maximum delay analysis.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KALmin

Specifies the KALmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KALmin** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment A of each timing check to obtain the adjustment value for that segment, for minimum delay analysis.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KALpctmax

Specifies the KALpctmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KALpctmax** variable specifies the percentage increment for maximum delay analysis for path segment A. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment A to calculate a modification of the scale factor specified by the **pbsa_KAmax** variable. The adjusted scale factor for path segment A is computed as follows:

```
Amax = MAX (pbsa_KAfloormax,  pbsa_KAmax+(numAlevels*pbsa_KALpctmax))
```

where the new factor Amax replaces the **pbsa_KAmax** variable in calculating delay adjustments.

The legal values for the **pbsa_KALpctmax** variable are between -1.0 and 0.0. A negative value causes a reduction of the positive KAmax adjustment factor. When the **pbsa_KALpctmax** variable is non-zero, the **pbsa_KALmax** variable is not used.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KALpctmin

Specifies the KALpctmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KALpctmin** variable specifies the percentage increment for minimum delay analysis for path segment A. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment A to calculate a modification of the scale factor specified by the **pbsa_KAmin** variable. The adjusted scale factor for path segment A is computed as follows:

```
Amin = MIN (pbsa_KAceilingmin,  pbsa_KAmin+(numAlevels*pbsa_KALpctmin))
```

where the new factor Amin replaces the **pbsa_KAmin** variable in calculating delay adjustments.

The legal values for the **pbsa_KALpctmin** variable are between 0.0 and 1.0. A positive value causes an increase in the negative KAmin adjustment factor (the factor becomes less negative). When the **pbsa_KALpctmin** variable is non-zero, the **pbsa_KALmin** variable is not used.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KAceilingmin

Specifies the limit of adjustment that can be made using the **pbsa_KALpctmin** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KALpctmin** variable to modify the derating factor of path segment A for minimum delay analysis, the **pbsa_KAceilingmin** variable sets an upper limit on the new factor. The adjusted value is computed as follows:

```
Amin = MIN (pbsa_KAceilingmin,  pbsa_KAmin+(numAlevels*pbsa_KALpctmin))
```

where the new factor Amin replaces the **pbsa_KAmin** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KAmin < pbsa_KAceilingmin < 0.0
```

If the **pbsa_KAceilingmin** variable is less than the **pbsa_KAmin** variable, the **pbsa_KAmin** variable value is used and no adjustment is made.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KAfloormax

Specifies the limit of adjustment that can be made using the **pbsa_KALpctmax** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KALpctmax** variable to modify the derating factor of path segment A for maximum delay analysis, the **pbsa_KAfloormax** variable sets a lower limit on the new factor. The adjusted value is computed as follows:

```
Amax = MAX (pbsa_KAfloormax,  pbsa_KAmax+(numAlevels*pbsa_KALpctmax))
```

where the new factor Amax replaces the **pbsa_KAmax** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KAmax > pbsa_KAfloormax > 0.0
```

If the **pbsa_KAfloormax** variable is greater than the **pbsa_KAmax** variable, the **pbsa_KAmax** variable value is used and no adjustment is made.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KAmax

Specifies the KAmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KAmax** variable specifies the KAmax scaling factor. NanoTime multiplies this factor by the original delay of segment A to obtain the adjustment value for that segment, for maximum delay analysis.

The legal values for the **pbsa_KAmax** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KAmin

Specifies the KAmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KAmin** variable specifies the KAmin scaling factor. NanoTime multiplies this factor by the original delay of segment A to obtain the adjustment value for that segment, for minimum delay analysis.

The legal values for the **pbsa_KAmin** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KAwiremax

Specifies the KAwiremax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KAwiremax** variable specifies the KAwiremax scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment A to obtain the adjustment value for that segment, for maximum delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KAwiremax** variable are betweeen 0.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KAwiremin

Specifies the KAwiremin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KAwiremin** variable specifies the KAwiremin scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment A to obtain the adjustment value for that segment, for min delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KAwiremin** variable are betweeen -1.0 and 0.0, inclusive.

The default is zero, which means that no adjustment is done. The variables for segment A are typically set to zero because segment A (the common segment) does not contribute to the uncertainty of a timing check.

Other variables also affect the segment A delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KBLmax

Specifies the KBLmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBLmax** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment B of each timing check to obtain the adjustment value for that segment, for maximum delay analysis.

The default is zero, which means that no adjustment is done. A positive value results in a smaller or more negative reported slack.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBmax(3)
pbsa_KBLmin(3)
pbsa_KBLpctmin(3)
pbsa_KBLpctmax(3)
pbsa_KBceilingmin(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBLmin

Specifies the KBLmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBLmin** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment B of each timing check to obtain the adjustment value for that segment, for minimum delay analysis.

The default is zero, which means that no adjustment is done. A negative value results in a smaller or more negative reported slack.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBmax(3)
pbsa_KBLmax(3)
pbsa_KBLpctmin(3)
pbsa_KBLpctmax(3)
pbsa_KBceilingmin(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.

pbsa_KBLpctmax

Specifies the KBLpctmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBLpctmax** variable specifies the percentage increment for maximum delay analysis for path segment B. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment B to calculate a modification of the scale factor specified by the **pbsa_KBmax** variable. The adjusted scale factor for path segment B is computed as follows:

```
Bmax = MAX (pbsa_KBfloormax,  pbsa_KBmax+(numBlevels*pbsa_KBLpctmax))
```

where the new factor Bmax replaces the **pbsa_KBmax** variable in calculating delay adjustments.

The legal values for the **pbsa_KBLpctmax** variable are between -1.0 and 0.0. A negative value causes a reduction of the positive KBmax adjustment factor. When the **pbsa_KBLpctmax** variable is non-zero, the **pbsa_KBLmax** variable is not used.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBmax(3)
pbsa_KBLmax(3)
pbsa_KBLmin(3)
pbsa_KBLpctmin(3)
pbsa_KBceilingmin(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBLpctmin

Specifies the KBLpctmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBLpctmin** variable specifies the percentage increment for minimum delay analysis for path segment B. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment B to calculate a modification of the scale factor specified by the **pbsa_KBmin** variable. The adjusted scale factor for path segment B is computed as follows:

```
Bmin = MIN (pbsa_KBceilingmin,  pbsa_KBmin+(numBlevels*pbsa_KBLpctmin))
```

where the new factor Bmin replaces the **pbsa_KBmin** variable in calculating delay adjustments.

The legal values for the **pbsa_KBLpctmin** variable are between 0.0 and 1.0. A positive value causes an increase in the negative KBmin adjustment factor (the factor becomes less negative). When the **pbsa_KBLpctmin** variable is non-zero, the **pbsa_KBLmin** variable is not used.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBmax(3)
pbsa_KBLmax(3)
pbsa_KBLmin(3)
pbsa_KBLpctmax(3)
pbsa_KBceilingmin(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBceilingmin

Specifies the limit of adjustment that can be made using the **pbsa_KBLpctmin** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KBLpctmin** variable to modify the derating factor of path segment B for minimum delay analysis, the **pbsa_KBceilingmin** variable sets an upper limit on the new factor. The adjusted value is computed as follows:

```
Bmin = MIN (pbsa_KBceilingmin,  pbsa_KBmin+(numBlevels*pbsa_KBLpctmin))
```

where the new factor Bmin replaces the **pbsa_KBmin** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KBmin < pbsa_KBceilingmin < 0.0
```

If the **pbsa_KBceilingmin** variable is less than the **pbsa_KBmin** variable, the **pbsa_KBmin** variable value is used and no adjustment is made.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBmax(3)
pbsa_KBLmin(3)
pbsa_KBLmax(3)
pbsa_KBLpctmin(3)
pbsa_KBLpctmax(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBfloormax

Specifies the limit of adjustment that can be made using the **pbsa_KBLpctmax** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KBLpctmax** variable to modify the derating factor of path segment B for maximum delay analysis, the **pbsa_KBfloormax** variable sets a lower limit on the new factor. The adjusted value is computed as follows:

```
Bmax = MAX (pbsa_KBfloormax,  pbsa_KBmax+(numAlevels*pbsa_KBLpctmax))
```

where the new factor Bmax replaces the **pbsa_KBmax** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KBmax > pbsa_KBfloormax > 0.0
```

If the **pbsa_KBfloormax** variable is greater than the **pbsa_KBmax** variable, the **pbsa_KBmax** variable value is used and no adjustment is made.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBmax(3)
pbsa_KBLmin(3)
pbsa_KBLmax(3)
pbsa_KBLpctmin(3)
pbsa_KBLpctmax(3)
pbsa_KBceilingmin(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBmax

Specifies the KBmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBmax** variable specifies the KBmax scaling factor. NanoTime multiplies this factor by the original delay of segment B to obtain the adjustment value for that segment, for maximum delay analysis.

The legal values for the **pbsa_KBmax** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. A positive value results in a smaller or more negative reported slack.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmin(3)
pbsa_KBLmax(3)
pbsa_KBLmin(3)
pbsa_KBLpctmin(3)
pbsa_KBLpctmax(3)
pbsa_KBceilingmin(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBmin

Specifies the KBmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBmin** variable specifies the KBmin scaling factor. NanoTime multiplies this factor by the original delay of segment B to obtain the adjustment value for that segment, for minimum delay analysis.

The legal values for the **pbsa_KBmin** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. A negative value results in a smaller or more negative reported slack.

Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KBmax(3)
pbsa_KBLmax(3)
pbsa_KBLmin(3)
pbsa_KBLpctmin(3)
pbsa_KBLpctmax(3)
pbsa_KBceilingmin(3)
pbsa_KBfloormax(3)

Similar pbsa_K* variables exist for paths A, C, and D.
```

pbsa_KBwiremax

Specifies the KBwiremax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBwiremax** variable specifies the KBwiremax scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment B to obtain the adjustment value for that segment, for maximum delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KBwiremax** variable are betweeen 0.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.

pbsa_KBwiremin

Specifies the KBwiremin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KBwiremin** variable specifies the KBwiremin scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment B to obtain the adjustment value for that segment, for min delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KBwiremin** variable are betweeen -1.0 and 0.0, inclusive.

The default is zero, which means that no adjustment is done. Other variables also affect the segment B delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KCLmax

Specifies the KCLmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCLmax** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment C of each timing check to obtain the adjustment value for that segment, for maximum delay analysis.

The default is zero, which means that no adjustment is done. A positive value results in a smaller or more negative reported slack.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCmax(3)
pbsa_KCLmin(3)
pbsa_KCLpctmin(3)
pbsa_KCLpctmax(3)
pbsa_KCceilingmin(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.

pbsa_KCLmin

Specifies the KCLmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCLmin** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment C of each timing check to obtain the adjustment value for that segment, for minimum delay analysis.

The default is zero, which means that no adjustment is done. A negative value results in a smaller or more negative reported slack.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCmax(3)
pbsa_KCLmax(3)
pbsa_KCLpctmin(3)
pbsa_KCLpctmax(3)
pbsa_KCceilingmin(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCLpctmax

Specifies the KCLpctmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCLpctmax** variable specifies the percentage increment for maximum delay analysis for path segment C. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment C to calculate a modification of the scale factor specified by the **pbsa_KCmax** variable. The adjusted scale factor for path segment C is computed as follows:

```
Cmax = MAX (pbsa_KCfloormax,  pbsa_KCmax+(numClevels*pbsa_KCLpctmax))
```

where the new factor Cmax replaces the **pbsa_KCmax** variable in calculating delay adjustments.

The legal values for the **pbsa_KCLpctmax** variable are between -1.0 and 0.0. A negative value causes a reduction of the positive KCmax adjustment factor. When the **pbsa_KCLpctmax** variable is non-zero, the **pbsa_KCLmax** variable is not used.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCmax(3)
pbsa_KCLmin(3)
pbsa_KCLmax(3)
pbsa_KCLpctmin(3)
pbsa_KCceilingmin(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCLpctmin

Specifies the KCLpctmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCLpctmin** variable specifies the percentage increment for minimum delay analysis for path segment C. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment C to calculate a modification of the scale factor specified by the **pbsa_KCmin** variable. The adjusted scale factor for path segment C is computed as follows:

```
Cmin = MIN (pbsa_KCceilingmin,  pbsa_KCmin+(numClevels*pbsa_KCLpctmin))
```

where the new factor Cmin replaces the **pbsa_KCmin** variable in calculating delay adjustments.

The legal values for the **pbsa_KCLpctmin** variable are between 0.0 and 1.0. A positive value causes an increase in the negative KCmin adjustment factor (the factor becomes less negative). When the **pbsa_KCLpctmin** variable is non-zero, the **pbsa_KCLmin** variable is not used.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCmax(3)
pbsa_KCLmin(3)
pbsa_KCLmax(3)
pbsa_KCLpctmax(3)
pbsa_KCceilingmin(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCceilingmin

Specifies the limit of adjustment that can be made using the **pbsa_KCLpctmin** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KCLpctmin** variable to modify the derating factor of path segment C for minimum delay analysis, the **pbsa_KCceilingmin** variable sets an upper limit on the new factor. The adjusted value is computed as follows:

```
Cmin = MIN (pbsa_KCceilingmin,  pbsa_KCmin+(numClevels*pbsa_KCLpctmin))
```

where the new factor Cmin replaces the **pbsa_KCmin** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KCmin < pbsa_KCceilingmin < 0.0
```

If the **pbsa_KCceilingmin** variable is less than the **pbsa_KCmin** variable, the **pbsa_KCmin** variable value is used and no adjustment is made.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCmax(3)
pbsa_KCLmin(3)
pbsa_KCLmax(3)
pbsa_KCLpctmin(3)
pbsa_KCLpctmax(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCfloormax

Specifies the limit of adjustment that can be made using the **pbsa_KCLpctmax** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KCLpctmax** variable to modify the derating factor of path segment C for maximum delay analysis, the **pbsa_KCfloormax** variable sets a lower limit on the new factor. The adjusted value is computed as follows:

```
Cmax = MAX (pbsa_KCfloormax,  pbsa_KCmax+(numClevels*pbsa_KCLpctmax))
```

where the new factor Cmax replaces the **pbsa_KCmax** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KCmax > pbsa_KCfloormax > 0.0
```

If the **pbsa_KCfloormax** variable is greater than the **pbsa_KCmax** variable, the **pbsa_KCmax** variable value is used and no adjustment is made.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCmax(3)
pbsa_KCLmin(3)
pbsa_KCLmax(3)
pbsa_KCLpctmin(3)
pbsa_KCLpctmax(3)
pbsa_KCceilingmin(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCmax

Specifies the KCmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCmax** variable specifies the KCmax scaling factor. NanoTime multiplies this factor by the original delay of segment C to obtain the adjustment value for that segment, for maximum delay analysis.

The legal values for the **pbsa_KCmax** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. A positive value results in a smaller or more negative reported slack.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmin(3)
pbsa_KCLmin(3)
pbsa_KCLmax(3)
pbsa_KCLpctmin(3)
pbsa_KCLpctmax(3)
pbsa_KCceilingmin(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCmin

Specifies the KCmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCmin** variable specifies the KCmin scaling factor. NanoTime multiplies this factor by the original delay of segment C to obtain the adjustment value for that segment, for minimum delay analysis.

The legal values for the **pbsa_KCmin** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. A negative value results in a smaller or more negative reported slack.

Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KCmax(3)
pbsa_KCLmin(3)
pbsa_KCLmax(3)
pbsa_KCLpctmin(3)
pbsa_KCLpctmax(3)
pbsa_KCceilingmin(3)
pbsa_KCfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and D.
```

pbsa_KCwiremax

Specifies the KCwiremax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCwiremax** variable specifies the KCwiremax scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment C to obtain the adjustment value for that segment, for maximum delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KCwiremax** variable are betweeen 0.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.

pbsa_KCwiremin

Specifies the KCwiremin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KCwiremin** variable specifies the KCwiremin scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment C to obtain the adjustment value for that segment, for min delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KCwiremin** variable are betweeen -1.0 and 0.0, inclusive.

The default is zero, which means that no adjustment is done. Other variables also affect the segment C delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KDLmax

Specifies the KDLmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDLmax** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment D of each timing check to obtain the adjustment value for that segment, for maximum delay analysis.

The default is zero, which means that no adjustment is done. A positive value results in a smaller or more negative reported slack.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDmax(3)
pbsa_KDLmin(3)
pbsa_KDLpctmin(3)
pbsa_KDLpctmax(3)
pbsa_KDceilingmin(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.

pbsa_KDLmin

Specifies the KDLmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDLmin** variable specifies uncertainty per simulation level in units of ps per level. NanoTime multiplies this amount of time by the number of simulation levels in segment D of each timing check to obtain the adjustment value for that segment, for minimum delay analysis.

The default is zero, which means that no adjustment is done. A negative value results in a smaller or more negative reported slack.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDmax(3)
pbsa_KDLmax(3)
pbsa_KDLpctmin(3)
pbsa_KDLpctmax(3)
pbsa_KDceilingmin(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDLpctmax

Specifies the KDLpctmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDLpctmax** variable specifies the percentage increment for maximum delay analysis for path segment D. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment D to calculate a modification of the scale factor specified by the **pbsa_KDmax** variable. The adjusted scale factor for path segment D is computed as follows:

```
Dmax = MAX (pbsa_KDfloormax,  pbsa_KDmax+(numDlevels*pbsa_KDLpctmax))
```

where the new factor Dmax replaces the **pbsa_KDmax** variable in calculating delay adjustments.

The legal values for the **pbsa_KDLpctmax** variable are between -1.0 and 0.0. A negative value causes a reduction of the positive KDmax adjustment factor. When the **pbsa_KDLpctmax** variable is non-zero, the **pbsa_KDLmax** variable is not used.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDmax(3)
pbsa_KDLmin(3)
pbsa_KDLmax(3)
pbsa_KDLpctmin(3)
pbsa_KDceilingmin(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDLpctmin

Specifies the KDLpctmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDLpctmin** variable specifies the percentage increment for minimum delay analysis for path segment D. NanoTime multiplies this incremental percentage value by the number of simulation levels in path segment D to calculate a modification of the scale factor specified by the **pbsa_KDmin** variable. The adjusted scale factor for path segment D is computed as follows:

```
Dmin = MIN (pbsa_KDceilingmin,  pbsa_KDmin+(numDlevels*pbsa_KDLpctmin))
```

where the new factor Dmin replaces the **pbsa_KDmin** variable in calculating delay adjustments.

The legal values for the **pbsa_KDLpctmin** variable are between 0.0 and 1.0. A positive value causes an increase in the negative KDmin adjustment factor (the factor becomes less negative). When the **pbsa_KDLpctmin** variable is non-zero, the **pbsa_KDLmin** variable is not used.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDmax(3)
pbsa_KDLmin(3)
pbsa_KDLmax(3)
pbsa_KDLpctmax(3)
pbsa_KDceilingmin(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDceilingmin

Specifies the limit of adjustment that can be made using the **pbsa_KDLpctmin** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KDLpctmin** variable to modify the derating factor of path segment D for minimum delay analysis, the **pbsa_KDceilingmin** variable sets an upper limit on the new factor. The adjusted value is computed as follows:

```
Dmin = MIN (pbsa_KDceilingmin,  pbsa_KDmin+(numDlevels*pbsa_KDLpctmin))
```

where the new factor Dmin replaces the **pbsa_KDmin** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KDmin < pbsa_KDceilingmin < 0.0
```

If the **pbsa_KDceilingmin** variable is less than the **pbsa_KDmin** variable, the **pbsa_KDmin** variable value is used and no adjustment is made.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDmax(3)
pbsa_KDLmin(3)
pbsa_KDLmax(3)
pbsa_KDLpctmin(3)
pbsa_KDLpctmax(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDfloormax

Specifies the limit of adjustment that can be made using the **pbsa_KDLpctmax** variable.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

If you use the **pbsa_KDLpctmax** variable to modify the derating factor of path segment D for maximum delay analysis, the **pbsa_KDfloormax** variable sets a lower limit on the new factor. The adjusted value is computed as follows:

```
Dmax = MAX (pbsa_KDfloormax,  pbsa_KDmax+(numDlevels*pbsa_KDLpctmax))
```

where the new factor Dmax replaces the **pbsa_KDmax** variable in calculating delay adjustments.

The expected range of this variable is

```
pbsa_KDmax > pbsa_KDfloormax > 0.0
```

If the **pbsa_KDfloormax** variable is greater than the **pbsa_KDmax** variable, the **pbsa_KDmax** variable value is used and no adjustment is made.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDmax(3)
pbsa_KDLmin(3)
pbsa_KDLmax(3)
pbsa_KDLpctmin(3)
pbsa_KDLpctmax(3)
pbsa_KDceilingmin(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDmax

Specifies the KDmax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDmax** variable specifies the KDmax scaling factor. NanoTime multiplies this factor by the original delay of segment D to obtain the adjustment value for that segment, for maximum delay analysis.

The legal values for the **pbsa_KDmax** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. A positive value results in a smaller or more negative reported slack.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmin(3)
pbsa_KDLmin(3)
pbsa_KDLmax(3)
pbsa_KDLpctmin(3)
pbsa_KDLpctmax(3)
pbsa_KDceilingmin(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDmin

Specifies the KDmin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDmin** variable specifies the KDmin scaling factor. NanoTime multiplies this factor by the original delay of segment D to obtain the adjustment value for that segment, for minimum delay analysis.

The legal values for the **pbsa_KDmin** variable are betweeen -1.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. A negative value results in a smaller or more negative reported slack.

Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the

man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KDmax(3)
pbsa_KDLmin(3)
pbsa_KDLmax(3)
pbsa_KDLpctmin(3)
pbsa_KDLpctmax(3)
pbsa_KDceilingmin(3)
pbsa_KDfloormax(3)

Similar pbsa_K* variables exist for paths A, B, and C.
```

pbsa_KDwiremax

Specifies the KDwiremax adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDwiremax** variable specifies the KDwiremax scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment D to obtain the adjustment value for that segment, for maximum delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KDwiremax** variable are betweeen 0.0 and +1.0, inclusive.

The default is zero, which means that no adjustment is done. Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KDwiremin

Specifies the KDwiremin adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KDwiremin** variable specifies the KDwiremin scaling factor. NanoTime multiplies this factor by the original wire delay of arcs on segment D to obtain the adjustment value for that segment, for min delay analysis. This variable is only used if timing_pbsa_gate_delay_only is set to true

The legal values for the **pbsa_KDwiremin** variable are betweeen -1.0 and 0.0, inclusive.

The default is zero, which means that no adjustment is done. Other variables also affect the segment D delay. For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
timing_pbsa_gate_delay_only(3)
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KUhold

Specifies the KUhold adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

1

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KUhold** variable specifies the KUhold scaling factor. NanoTime multiplies this scaling factor by the hold uncertainty value set with the **set_clock_uncertainty** command to obtain the global uncertainty value used for hold analysis.

Leave this variable set to 1.0 if you want to use the **set_clock_uncertainty** value for hold checks (for example, to support reduced global uncertainty settings to account for off-chip clock uncertainty). Set this variable to 0.0 if you want path-based slack adjustment to entirely replace the global uncertainty setting for hold checks.

For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_KUsetup

Specifies the KUsetup adjustment factor for path-based slack adjustment.

TYPE

float

DEFAULT

1

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

The **pbsa_KUsetup** variable specifies the KUsetup scaling factor. NanoTime multiplies this scaling factor by the setup uncertainty value set with the **set_clock_uncertainty** command to obtain the global uncertainty value used for setup analysis.

Leave this variable set to 1.0 if you want to use the **set_clock_uncertainty** value for setup checks (for example, to support reduced global uncertainty settings to account for off-chip clock uncertainty). Set this variable to 0.0 if you want path-based slack adjustment to entirely replace the global uncertainty setting for setup checks.

For more information, see the NanoTime User Guide, the man pages for the **report_pbsa_calculation** command, and the man pages for other pbsa_* variables.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_allow_reconvergent_common_net

Allows selection of the common net in path-based slack adjustment analysis to be based on reconvergence.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default the common net is the net on the data and clock paths when they first diverge. When the **pbsa_allow_reconvergent_common_net** variable is set to **true**, the common net can occur beyond the first divergence if the data and clock paths converge later. This can lead to more pessimism removal provided the extended segment A is realistic for the design.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.

pbsa_common_net_use_same_direction_delays

When the common net edge directions for the data and clock paths differ, this variable results in the minimum of the common net differences between the data-rise arrival time and clock-rise arrival time and the data-fall arrival time and clock-fall arrival time to be credited back to the slack.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default the difference between the data segment A delay and the clock A segment delay is credited back to the slack. When the **pbsa_common_net_use_same_direction_delays** variable is set to **true**, the minimum of the differences between the data-rise arrival time and the clock-rise arrival time and the data-fall arrival time and the clock-fall arrival time for the common segment will be credited back to the slack.

For more information about path-based slack adjustment, see the man page for the

report_pbsa_calculation command or the NanoTime User Guide.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_include_common_si_deltas(3)
pbsa_same_edge_reconvergence_only(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_differential_switching_direction

Specifies how the switching direction of the differential nets on the clock and data paths should be considered when determining the common nets used during path-based slack adjustment calculations.

TYPE

string

DEFAULT

ignore

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in the PBSA variables modify the calculated path delays for each segment type.

By default, the arcs of segment A are considered common if they drive a differential net pair that switch in any direction. This behavior for differential nets can be altered by setting the **pbsa_differential_switching_direction** variable to **opposite**, **same**, or **automatic**. Selecting **automatic** causes NanoTime to infer the switching direction of the common net pair based on the timing check.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

report_pbsa_calculation(2)
pbsa_allow_reconvergent_common_net(3)
pbsa_same_common_switching_direction(3)

pbsa_enable_chained_generated_clocks

Specifies whether to include chained generated clock segments in the clock and data paths when determining the common net during path-based slack adjustment calculations.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default, NanoTime includes chained generated clock segments in both the clock and data paths when determining the common segment A.

Set the **pbsa_enable_chained_generated_clocks** variable to **false** if you want to save an extracted timing model with delays that are reported with respect to a point within the generated clock chain.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.

pbsa_include_common_si_deltas

This variable limits the removal of SI delta delays on the common data and clock paths to zero-cycle paths.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default all the SI delta delays for the data and clock A segments are included in the common mode adjustment. When the **pbsa_include_common_si_deltas** variable is set to **true**, the SI delta delays for the data and clock A segments will be considered common only for zero-cycle paths.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_same_edge_reconvergence_only(3)
pbsa_common_net_use_same_direction_delays(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.

pbsa_min_threshold

Specifies the minimum time threshold for using path-based slack adjustment values.

TYPE

float

DEFAULT

0

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

You can choose to ignore delay adjustments below a threshold value if you use the **-selective_pbsa** option of the **set_timing_check_attributes** command. The **pbsa_min_threshold** variable sets the minimum threshold for using the delay adjustment values. For example, if you set the **pbsa_min_threshold** variable to 0.1, if the overall path-based slack adjustment value for a path is less than 0.1 time units, the adjustment value is ignored.

The default is 0.0, causing all path-based slack adjustment values to be used, no matter how small.

This variable takes effect only if you specify the **-selective_pbsa** option of the

set_timing_check_attributes command.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

```
report_pbsa_calculation(2)
set_timing_check_attributes(2)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_same_common_switching_direction

Specifies whether the switching direction of the nets on the clock and data path should be considered when determining the common net used during path-based slack adjustment calculations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default the arcs of segment A are considered common if they drive the same nets. The switching direction is not considered. When the **pbsa_same_common_switching_direction** variable is set to **true**, the path arcs are not considered common unless they drive the same net and have the same switching direction.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.

pbsa_same_edge_reconvergence_only

When the common net edge directions for the data and clock paths differ at a reconvergent common net, this variable forces the common net to be the previous divergent net between the data and clock paths.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default the common net is the net on the data and clock paths when they first diverge. When the **pbsa_allow_reconvergent_common_net** variable is set to **true**, the common net can occur beyond the first divergence if the data and clock paths converge later. This can lead to more pessimism removal provided the extended segment A is realistic for the design. When the **pbsa_same_edge_reconvergence_only** variable is set to true and the data and clock paths have different edges at the reconvergent net, the previous divergent net is set as the common net. Note that the variable **pbsa_allow_reconvergent_common_net** must be true to enable the functionality of the

variable **pbsa_same_edge_reconvergence_only**.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_include_common_si_deltas(3)
pbsa_common_net_use_same_direction_delays(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pbsa_same_edge_zero_cycle_reconvergence_o

Restricts the same edge reconvergent common net analysis only to zero cycle paths. Even when the common net edge directions for the data and clock paths are the same at a reconvergent common net, this variable forces the common net to be the previous divergent net between the data and clock paths for non zero cycle paths.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You can use this feature to model clock uncertainty or device property variations, or to add custom timing margin or delay adjustments. You must have a NanoTime Ultra license to use path-based slack adjustment.

You invoke path-based slack adjustment by using the **-pbsa** option with the **trace_paths** command. Each path segment is classified as type A (from the external reference clock to the common point), B (from the common point to the clock pin of the launching sequential element), C (from the common point to the clock pin of the capturing sequential element), or D (from the clock pin of the launch element to the data input pin of the capture element). Factors that you set in user variables modify the calculated path delays for each segment type.

By default the common net is the net on the data and clock paths when they first diverge. When the **pbsa_allow_reconvergent_common_net** variable is set to **true**, the common net can occur beyond the first divergence if the data and clock paths converge later. This can lead to more pessimism removal provided the extended segment A is realistic for the design. When the **pbsa_same_edge_reconvergence_only** variable is set to true and the data and clock paths have

different edges at the reconvergent net, the previous divergent net is set as the common net. When the **pbsa_same_edge_zero_cycle_reconvergence_only** variable is set to true and the data and clock paths have same edges at the reconvergent net, but the paths are part of a 1+ cycle check, the previous divergent net is set as the common net. Note that both variables **pbsa_allow_reconvergent_common_net** and **pbsa_same_edge_reconvergence_only** must be true to enable the functionality of the variable **pbsa_same_edge_zero_cycle_reconvergence_only**.

For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command or the NanoTime User Guide.

SEE ALSO

```
report_pbsa_calculation(2)
pbsa_min_threshold(3)
pbsa_allow_reconvergent_common_net(3)
pbsa_enable_chained_generated_clocks(3)
pbsa_same_common_switching_direction(3)
pbsa_include_common_si_deltas(3)
pbsa_common_net_use_same_direction_delays(3)
pbsa_same_edge_reconvergence_only(3)
pbsa_KUsetup(3)
pbsa_KUhold(3)
pbsa_KAmin(3)
pbsa_KAmax(3)
pbsa_KALmin(3)
pbsa_KALmax(3)
pbsa_KALpctmin(3)
pbsa_KALpctmax(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

Similar pbsa_K* variables exist for paths B, C, and D.
```

pexe_wait_for_exit

Causes NanoTime to wait for all spawned processes to terminate when the exit, quit or suspend_licenses command is issued.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to **true**, NanoTime waits for all spawned processes to terminate when the **exit**, **quit**, or **suspend_licenses** command is invoked. If the variable is set to **false**, NanoTime errors out of the command being executed in the spawned process and returns control back to the session. The **false** setting is appropriate for interactive usage of NanoTime.

SEE ALSO

parallel_execute(2)
redirect(2)

port_search_in_current_instance

Controls whether the **get_ports** command can get ports of the current instance.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), the **get_ports** command gets ports of the current design. If this variable is set to **true**, the **get_ports** command gets ports of the current instance.

SEE ALSO

`get_ports(2)`

rc_behavioral_resistor_controlling_voltage

Specifies the controlling voltage for the conversion of a behavioral resistor into a parasitic resistor.

TYPE

float

DEFAULT

-1.0

DESCRIPTION

When NanoTime converts a behavioral resistor into a parasitic resistor, the value specified by this variable is used as the controlling voltage.

If you leave the **rc_behavioral_resistor_controlling_voltage** variable at its default of -1.0, the **link_design** command sets the controlling voltage to the value of the **oc_global_voltage** variable or to the default rail voltage if the **oc_global_voltage** variable is not set.

SEE ALSO

```
link_design(2)
rc_enable_behavioral_resistor_conversion(3)
oc_global_voltage(3)
```

rc_enable_behavioral_resistor_conversion

Enables the conversion of a behavioral resistor in the SPICE netlist into a single parasitic resistor during the **link_design** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime converts behavioral resistors into parasitic resistors during the design linking step. When the **rc_enable_behavioral_resistor_conversion** variable is set to **false**, NanoTime ignores all behavioral resistors.

SEE ALSO

```
link_design(2)
rc_behavioral_resistor_controlling_voltage(3)
PARS-105(n)
```

rc_input_threshold_full_transition

Specifies the interpretation of input transition time values, either as the time for the full rail-to-rail voltage swing or the time between the rc_slew_* threshold voltages.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable specifies the interpretation of input transition time values. If the variable is set to **true** (the default) NanoTime considers the input transition time values to be the amount of time it takes for the signal to change from 0% to 100% of the rail voltage for a rising transition, or from 100% to 0% of the rail voltage for a falling transition.

If the variable is set to **false**, NanoTime considers the input transition time values to be the amount of time it takes for the signal to change from the lower to upper thresholds for a rising transition, or from the upper to lower thresholds for a falling transition. The thresholds are set by the following variables:

```
Variable Name                     Default (percent)
-------------------------------------------------
rc_slew_lower_threshold_pct_fall         10
rc_slew_lower_threshold_pct_rise         10
rc_slew_upper_threshold_pct_fall         90
rc_slew_upper_threshold_pct_rise         90
```

SEE ALSO

```
rc_slew_derate_from_library(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_fall(3)
```

rc_input_threshold_pct_fall

Specifies the threshold that defines the time measurement point for falling-edge transitions at the startpoint of a delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

The **rc_input_threshold_pct_fall** variable specifies the threshold that defines the time measurement point for a falling-edge signal transition at the startpoint of a delay calculation. The value is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default is 50, which means that a falling-edge transition is considered to occur when the signal reaches the halfway point between the voltage rails.

This is one of eight variables that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values (transition times) and propagation delays in the presence of parasitics annotated with the **read_parasitics** command. The variables are as follows:

```
Variable Name                      Default (percent)
-------------------------------------------------
rc_input_threshold_pct_fall              50
rc_input_threshold_pct_rise              50
rc_output_threshold_pct_fall             50
rc_output_threshold_pct_rise             50
rc_slew_lower_threshold_pct_rise         10
rc_slew_lower_threshold_pct_fall         10
rc_slew_upper_threshold_pct_rise         90
rc_slew_upper_threshold_pct_fall         90
```

Using the defaults, NanoTime calculates delays starting from the point where the input transition crosses the 50% threshold and ending at the point where the output transition crosses the 50% threshold. To calculate slew for a rising transition, NanoTime uses the time from the 10% crossing point to the 90% crossing point; or for a falling transition, from the 90% crossing point to the 10% crossing point.

EXAMPLES

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 40% threshold to the point at which the output transition crosses the 75% threshold. To calculate transition times, NanoTime considers the time from the 20% to the 80% threshold crossing points for a rising transition, or from the 80% to the 20% threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_rise 40
nt_shell> set rc_input_threshold_pct_fall 40
nt_shell> set rc_output_threshold_pct_rise 75
nt_shell> set rc_output_threshold_pct_fall 75
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

read_parasitics(2)
report_annotated_parasitics(2)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)

rc_input_threshold_pct_rise

Specifies the threshold voltage that defines the time measurement point for rising-edge transitions at the startpoint of a delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

The **rc_input_threshold_pct_rise** variable specifies the threshold that defines the time measurement point for a rising-edge signal transition at the startpoint of a delay calculation. The value is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default is 50, which means that a rising-edge transition is considered to occur when the signal reaches the halfway point between the voltage rails.

This is one of eight variables that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values (transition times) and propagation delays in the presence of parasitics annotated with the **read_parasitics** command. The variables are as follows:

```
Variable Name                      Default (percent)
-------------------------------------------------
rc_input_threshold_pct_fall              50
rc_input_threshold_pct_rise              50
rc_output_threshold_pct_fall             50
rc_output_threshold_pct_rise             50
rc_slew_lower_threshold_pct_rise         10
rc_slew_lower_threshold_pct_fall         10
rc_slew_upper_threshold_pct_rise         90
rc_slew_upper_threshold_pct_fall         90
```

Using the defaults, NanoTime calculates delays starting from the point where the input transition crosses

the 50% threshold and ending at the point where the output transition crosses the 50% threshold. To calculate slew for a rising transition, NanoTime uses the time from the 10% crossing point to the 90% crossing point; or for a falling transition, from the 90% crossing point to the 10% crossing point.

EXAMPLES

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 40% threshold to the point at which the output transition crosses the 75% threshold. To calculate transition times, NanoTime considers the time from the 20% to the 80% threshold crossing points for a rising transition, or from the 80% to the 20% threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_rise 40
nt_shell> set rc_input_threshold_pct_fall 40
nt_shell> set rc_output_threshold_pct_rise 75
nt_shell> set rc_output_threshold_pct_fall 75
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

read_parasitics(2)
report_annotated_parasitics(2)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)

rc_output_threshold_pct_fall

Specifies the threshold voltage that defines the time measurement point for falling-edge transitions at the endpoint of a delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

The **rc_output_threshold_pct_fall** variable specifies the threshold that defines the time measurement point for a falling-edge signal transition at the endpoint of a delay calculation. The value is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default is 50, which means that a falling-edge transition is considered to occur when the signal reaches the halfway point between the voltage rails.

This is one of eight variables that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values (transition times) and propagation delays in the presence of parasitics annotated with the **read_parasitics** command. The variables are as follows:

```
Variable Name                      Default (percent)
-------------------------------------------------
rc_input_threshold_pct_fall              50
rc_input_threshold_pct_rise              50
rc_output_threshold_pct_fall             50
rc_output_threshold_pct_rise             50
rc_slew_lower_threshold_pct_rise         10
rc_slew_lower_threshold_pct_fall         10
rc_slew_upper_threshold_pct_rise         90
rc_slew_upper_threshold_pct_fall         90
```

Using the defaults, NanoTime calculates delays starting from the point where the input transition crosses the 50% threshold and ending at the point where the output transition crosses the 50% threshold. To calculate slew for a rising transition, NanoTime uses the time from the 10% crossing point to the 90% crossing point; or for a falling transition, from the 90% crossing point to the 10% crossing point.

EXAMPLES

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 40% threshold to the point at which the output transition crosses the 75% threshold. To calculate transition times, NanoTime considers the time from the 20% to the 80% threshold crossing points for a rising transition, or from the 80% to the 20% threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_rise 40
nt_shell> set rc_input_threshold_pct_fall 40
nt_shell> set rc_output_threshold_pct_rise 75
nt_shell> set rc_output_threshold_pct_fall 75
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

read_parasitics(2)
report_annotated_parasitics(2)
rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)

rc_output_threshold_pct_rise

Specifies the threshold voltage that defines the time measurement point for rising-edge transitions at the endpoint of a delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

The **rc_output_threshold_pct_rise** variable specifies the threshold that defines the time measurement point for a rising-edge signal transition at the endpoint of a delay calculation. The value is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default is 50, which means that a rising-edge transition is considered to occur when the signal reaches the halfway point between the voltage rails.

This is one of eight variables that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values (transition times) and propagation delays in the presence of parasitics annotated with the **read_parasitics** command. The variables are as follows:

```
Variable Name                      Default (percent)
-------------------------------------------------
rc_input_threshold_pct_fall              50
rc_input_threshold_pct_rise              50
rc_output_threshold_pct_fall             50
rc_output_threshold_pct_rise             50
rc_slew_lower_threshold_pct_rise         10
rc_slew_lower_threshold_pct_fall         10
rc_slew_upper_threshold_pct_rise         90
rc_slew_upper_threshold_pct_fall         90
```

Using the defaults, NanoTime calculates delays starting from the point where the input transition crosses

the 50% threshold and ending at the point where the output transition crosses the 50% threshold. To calculate slew for a rising transition, NanoTime uses the time from the 10% crossing point to the 90% crossing point; or for a falling transition, from the 90% crossing point to the 10% crossing point.

EXAMPLES

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 40% threshold to the point at which the output transition crosses the 75% threshold. To calculate transition times, NanoTime considers the time from the 20% to the 80% threshold crossing points for a rising transition, or from the 80% to the 20% threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_rise 40
nt_shell> set rc_input_threshold_pct_fall 40
nt_shell> set rc_output_threshold_pct_rise 75
nt_shell> set rc_output_threshold_pct_fall 75
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

read_parasitics(2)
report_annotated_parasitics(2)
rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)

rc_reduction_exclude_boundary_nets

Excludes boundary nets from RC reduction to support model extraction with full parasitics on the block boundary.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When you use the **read_parasitics** command, NanoTime simplifies (reduces) complex annotated RC parasitics to improve the analysis runtime and reduce the amount of data to analyze. The tool combines or removes RC sections from each net to make a simpler representation that has a path delay within a specified tolerance of the original RC network.

When the **rc_reduction_exclude_boundary_nets** variable is set to **true**, boundary nets are excluded from RC reduction, thereby retaining the full, original parasitics on the boundary. This is typically done in anticipation of using the **extract_model** command with the **-extract_boundary_parasitics** option, which generates a model that maintains the boundary parasitics.

The default is **false**, which causes all parasitics to be reduced, including the boundary parasitics.

SEE ALSO

`extract_model(2)`

```
read_parasitics(2)
rc_reduction_max_net_delta_delay(3)
```

rc_reduction_max_net_delta_delay

Sets the maximum time delay threshold for RC reduction, in nanoseconds.

TYPE

float

DEFAULT

0.01

DESCRIPTION

NanoTime reduces (simplifies) complex annotated RC parasitics for the purpose of limiting the amount of design data and optimizing the analysis runtime.

The tool analyzes the delay and performs reduction on a net-by-net basis. All RC elements on a net (known as the "RC cloud") are included. The RC cloud includes coupling capacitances on the net only if signal integrity analysis is enabled. In other words, NanoTime includes all the parasitic elements that make up the wire delay on a net in the parasitics reduction operation.

RC reduction considers a RC network time constant at the input of the RC cloud, therefore input transition times do not affect the reduction operation.

The reduction operation occurs at the **check_design** command. To successfully exit this phase of the flow, you must complete partially annotated nets by using some combination of the **complete_net_parasitics**, **read_parasitics -complete_with**, or **check_design -complete_with** commands. These commands affect the parasitics reduction results because they insert resistors and capacitors into the network. You specify the value of the inserted elements by setting the **parasitics_completion_capacitance** and **parasitics_completion_resistance** variables.

Variables that control analysis options are taken into account later in the NanoTime flow and therefore do not affect parasitic reduction. For example, variables for extended side branch analysis and Miller capacitance analysis do not affect the reduction operation.

During the parasitics reduction operation, NanoTime combines or removes RC sections from each net to make a simpler representation that has a delay within a tolerance of the original delay of that net. The change in delay can be no more than the value of the **rc_reduction_max_net_delta_delay** variable. If the change in delay is less than the value of the **rc_reduction_min_net_delta_delay** variable, no reduction occurs.

By default, these variables are set to 0.001 and 0.01 nanoseconds, respectively (1 and 10 picoseconds). The default tolerance is therefore a value between 1 and 10 picoseconds.

A larger delay tolerance allows more RC reduction, resulting in less runtime. Setting the **rc_reduction_max_net_delta_delay** variable to a large value such as 1.0 (1000 ps) results in maximum reduction, essentially getting rid of all resistors. On the other hand, setting the **rc_reduction_max_net_delta_delay** variable to zero prevents any parasitic reduction from occurring.

Runtime might be severely affected if RC reduction is either too strict or disabled completely. To achieve a good balance between accuracy and runtime for advanced technology nodes, set the variables as follows:

```
set rc_reduction_max_net_delta_delay 0.0005
set rc_reduction_min_net_delta_delay 0.00005
```

If the value of the **rc_reduction_max_net_delta_delay** variable setting is less than the value of the **rc_reduction_min_net_delta_delay** variable, NanoTime ignores the **rc_reduction_min_net_delta_delay** variable.

A small (or 0.0) value may be overidden by a larger value in an **PARS-024** message.

SEE ALSO

```
read_parasitics(2)
rc_reduction_min_net_delta_delay(3)
parasitics_completion_capacitance(3)
parasitics_completion_resistance(3)
complete_net_parasitics(2)
check_design(2)
PARS-024(n)
```

rc_reduction_min_net_delta_delay

Sets the minimum time delay threshold for RC reduction, in nanoseconds.

TYPE

float

DEFAULT

0.001

DESCRIPTION

NanoTime reduces (simplifies) complex annotated RC parasitics for the purpose of limiting the amount of design data and optimizing the analysis runtime.

The tool analyzes the delay and performs reduction on a net-by-net basis. All RC elements on a net (known as the "RC cloud") are included. The RC cloud includes coupling capacitances on the net only if signal integrity analysis is enabled. In other words, NanoTime includes all the parasitic elements that make up the wire delay on a net in the parasitics reduction operation.

RC reduction considers a RC network time constant at the input of the RC cloud, therefore input transition times do not affect the reduction operation.

The reduction operation occurs at the **check_design** command. To successfully exit this phase of the flow, you must complete partially annotated nets by using some combination of the **complete_net_parasitics**, **read_parasitics -complete_with**, or **check_design -complete_with** commands. These commands affect the parasitics reduction results because they insert resistors and capacitors into the network. You specify the value of the inserted elements by setting the **parasitics_completion_capacitance** and **parasitics_completion_resistance** variables.

Variables that control analysis options are taken into account later in the NanoTime flow and therefore do not affect parasitic reduction. For example, variables for extended side branch analysis and Miller capacitance analysis do not affect the reduction operation.

During the parasitics reduction operation, NanoTime combines or removes RC sections from each net to make a simpler representation that has a delay within a tolerance of the original delay of that net. The change in delay can be no more than the value of the **rc_reduction_max_net_delta_delay** variable. If the change in delay is less than the value of the **rc_reduction_min_net_delta_delay** variable, no reduction occurs.

By default, these variables are set to 0.001 and 0.01 nanoseconds, respectively (1 and 10 picoseconds). The default tolerance is therefore a value between 1 and 10 picoseconds.

A larger delay tolerance allows more RC reduction, resulting in less runtime. Setting the **rc_reduction_max_net_delta_delay** variable to a large value such as 1.0 (1000 ps) results in maximum reduction, essentially getting rid of all resistors. On the other hand, setting the **rc_reduction_max_net_delta_delay** variable to zero prevents any parasitic reduction from occurring.

Runtime might be severely affected if RC reduction is either too strict or disabled completely. To achieve a good balance between accuracy and runtime for advanced technology nodes, set the variables as follows:

```
set rc_reduction_max_net_delta_delay 0.0005
set rc_reduction_min_net_delta_delay 0.00005
```

If the value of the **rc_reduction_max_net_delta_delay** variable setting is less than the value of the **rc_reduction_min_net_delta_delay** variable, NanoTime ignores the **rc_reduction_min_net_delta_delay** variable.

A small (or 0.0) value may be overidden by a larger value in an **PARS-024** message.

SEE ALSO

```
read_parasitics(2)
rc_reduction_max_net_delta_delay(3)
parasitics_completion_capacitance(3)
parasitics_completion_resistance(3)
complete_net_parasitics(2)
check_design(2)
PARS-024(n)
```

rc_slew_derate_from_library

Specifies a derating factor for library-defined transition times.

TYPE

float

DEFAULT

1

DESCRIPTION

The **rc_slew_derate_from_library** variable specifies the derating factor needed for the transition times in the Synopsys library to match the characterization trip points used for analysis. The default is 1.0, meaning that the transition times in the Synopsys library are used without change. Set this variable only when using parasitic annotation.

To apply a derating factor to the library transition times, set the **rc_slew_derate_from_library** variable to a value between 0.0 and 1.0. NanoTime multiplies the library-defined transition times by this factor before using them for analysis.

There is no need to set the **rc_slew_derate_from_library** variable if the transition times specified in the library represent the exact transition times between the characterization trip points. However, you need to set this variable for libraries in which the transition times in the Synopsys library have been extrapolated to the rail voltages.

For example, if the transition times were characterized between 30 percent and 70 percent and then extrapolated to the rails (0 percent to 100 percent), the **rc_slew_derate_from_library** variable should be set to the ratio of the two spans, calculated as (0.70-0.30)/(1.00-0.00) = 0.40. NanoTime multiplies the library-specified slew values by 0.40, thus restoring the original slew values between the 30 percent and 70 percent trip points.

The following commands specify the threshold levels that were used to characterize cell delays, as defined

in the Synopsys library:

```
nt_shell> set rc_slew_derate_from_library 0.4
nt_shell> set rc_slew_lower_threshold_pct_fall 30
nt_shell> set rc_slew_lower_threshold_pct_rise 30
nt_shell> set rc_slew_upper_threshold_pct_fall 70
nt_shell> set rc_slew_upper_threshold_pct_rise 70
nt_shell> set rc_input_threshold_pct_rise 50
nt_shell> set rc_input_threshold_pct_fall 50
nt_shell> set rc_output_threshold_pct_rise 50
nt_shell> set rc_output_threshold_pct_fall 50
```

The four slew threshold variables specify that slew values were characterized by measuring the transition times from 30 to 70 percent and from 70 to 30 percent of the supply voltage. The slew derating variable, which is set to 0.4, specifies that the transition times were extrapolated to the rail voltages (0 to 100 percent of the supply voltage); the range of 30 to 70 percent is a span of 40 percent of the supply voltage. The two input threshold and two output threshold variables specify that delays were calculated from trip points at 50 percent of the supply voltage.

SEE ALSO

```
read_parasitics(2)
report_annotated_parasitics(2)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_input_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
```

rc_slew_lower_threshold_pct_fall

Specifies the threshold voltage that defines the time measurement endpoint of a falling transition, used for calculating the slew (transition time).

TYPE

float

DEFAULT

10

DESCRIPTION

This variable specifies the threshold voltage that defines the time measurement endpoint of a falling transition, used for calculating the slew (also known as transition time). The slew for a falling transition is the amount of time it takes for the signal to change from a high state to a low state.

The variable setting is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default setting is 10, which means that a falling transition is considered to end when the signal falls to the voltage that is 10 percent of the rail-to-rail voltage.

This is one of eight variables, listed below, that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values and propagation delays in the presence of parasitics annotated with the **read_parasitics** command.

Variable Name	Default
rc_input_threshold_pct_fall	50.0
rc_input_threshold_pct_rise	50.0
rc_output_threshold_pct_fall	50.0
rc_output_threshold_pct_rise	50.0
rc_slew_lower_threshold_pct_fall	10.0
rc_slew_lower_threshold_pct_rise	10.0
rc_slew_upper_threshold_pct_fall	90.0

```
rc_slew_upper_threshold_pct_rise  90.0
```

With the default settings, NanoTime calculates each delay starting from the point where the input transition crosses the 50 percent threshold, and ending at the point where the output transition crosses the 50 percent threshold. To calculate slew for a rising transition, NanoTime considers the time from the 10 percent crossing point to the 90 percent crossing point; or for a falling transition, from the 90 percent crossing point to the 10 percent crossing point.

To specify these parameters for a particular net rather than globally, use the **set_measurement_threshold** command.

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 50 percent threshold to the point at which the output transition crosses the 55 percent threshold (55 percent of the rail-to-rail voltage). Furthermore, to calculate transition times, NanoTime considers the time from the 20 percent to the 80 percent threshold crossing points for a rising transition, or from the 80 percent to the 20 percent threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_fall 50
nt_shell> set rc_input_threshold_pct_rise 50
nt_shell> set rc_output_threshold_pct_fall 55
nt_shell> set rc_output_threshold_pct_rise 55
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

```
read_parasitics(2)
report_annotated_parasitics(2)
set_measurement_threshold(2)
rc_input_threshold_full_transition(3)
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_lower_threshold_pct_rise

Specifies the threshold voltage that defines the time measurement startpoint of a rising transition, used for calculating the slew (transition time).

TYPE

float

DEFAULT

10

DESCRIPTION

This variable specifies the threshold voltage that defines the time measurement startpoint of a rising transition, used for calculating the slew (also known as transition time). The slew for a rising transition is the amount of time it takes for the signal to change from a low state to a high state.

The variable setting is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default setting is 10, which means that a rising transition is considered to start when the signal crosses the voltage level that is 10 percent of the rail-to-rail voltage.

This is one of eight variables, listed below, that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values and propagation delays in the presence of parasitics annotated with the **read_parasitics** command.

Variable Name	Default
rc_input_threshold_pct_fall	50.0
rc_input_threshold_pct_rise	50.0
rc_output_threshold_pct_fall	50.0
rc_output_threshold_pct_rise	50.0
rc_slew_lower_threshold_pct_fall	10.0
rc_slew_lower_threshold_pct_rise	10.0
rc_slew_upper_threshold_pct_fall	90.0

```
rc_slew_upper_threshold_pct_rise  90.0
```

With the default settings, NanoTime calculates each delay starting from the point where the input transition crosses the 50 percent threshold, and ending at the point where the output transition crosses the 50 percent threshold. To calculate slew for a rising transition, NanoTime considers the time from the 10 percent crossing point to the 90 percent crossing point; or for a falling transition, from the 90 percent crossing point to the 10 percent crossing point.

To specify these parameters for a particular net rather than globally, use the **set_measurement_threshold** command.

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 50 percent threshold to the point at which the output transition crosses the 55 percent threshold (55 percent of the rail-to-rail voltage). Furthermore, to calculate transition times, NanoTime considers the time from the 20 percent to the 80 percent threshold crossing points for a rising transition, or from the 80 percent to the 20 percent threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_fall 50
nt_shell> set rc_input_threshold_pct_rise 50
nt_shell> set rc_output_threshold_pct_fall 55
nt_shell> set rc_output_threshold_pct_rise 55
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

```
read_parasitics(2)
report_annotated_parasitics(2)
set_measurement_threshold(2)
rc_input_threshold_full_transition(3)
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_upper_threshold_pct_fall

Specifies the threshold voltage that defines the time measurement startpoint of a falling transition, used for calculating the slew (transition time).

TYPE

float

DEFAULT

90

DESCRIPTION

This variable specifies the threshold voltage that defines the time measurement startpoint of a falling transition, used for calculating the slew (also known as transition time). The slew for a falling transition is the amount of time it takes for the signal to change from a high state to a low state.

The variable setting is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default setting is 90, which means that a falling transition is considered to start when the signal crosses the voltage level that is 90 percent of the rail-to-rail voltage.

This is one of eight variables, listed below, that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values and propagation delays in the presence of parasitics annotated with the **read_parasitics** command.

Variable Name	Default
rc_input_threshold_pct_fall	50.0
rc_input_threshold_pct_rise	50.0
rc_output_threshold_pct_fall	50.0
rc_output_threshold_pct_rise	50.0
rc_slew_lower_threshold_pct_fall	10.0
rc_slew_lower_threshold_pct_rise	10.0
rc_slew_upper_threshold_pct_fall	90.0

```
rc_slew_upper_threshold_pct_rise  90.0
```

With the default settings, NanoTime calculates each delay starting from the point where the input transition crosses the 50 percent threshold, and ending at the point where the output transition crosses the 50 percent threshold. To calculate slew for a rising transition, NanoTime considers the time from the 10 percent crossing point to the 90 percent crossing point; or for a falling transition, from the 90 percent crossing point to the 10 percent crossing point.

To specify these parameters for a particular net rather than globally, use the **set_measurement_threshold** command.

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 50 percent threshold to the point at which the output transition crosses the 55 percent threshold (55 percent of the rail-to-rail voltage). Furthermore, to calculate transition times, NanoTime considers the time from the 20 percent to the 80 percent threshold crossing points for a rising transition, or from the 80 percent to the 20 percent threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_fall 50
nt_shell> set rc_input_threshold_pct_rise 50
nt_shell> set rc_output_threshold_pct_fall 55
nt_shell> set rc_output_threshold_pct_rise 55
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

```
read_parasitics(2)
report_annotated_parasitics(2)
set_measurement_threshold(2)
rc_input_threshold_full_transition(3)
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_upper_threshold_pct_rise

Specifies the threshold voltage that defines the time measurement endpoint of a rising transition, used for calculating the slew (transition time).

TYPE

float

DEFAULT

90

DESCRIPTION

This variable specifies the threshold voltage that defines the time measurement endpoint of a rising transition, used for calculating the slew (also known as transition time). The slew for a rising transition is the amount of time it takes for the signal to change from a low state to a high state.

The variable setting is a percentage of the full rail-to-rail voltage swing, a number from 0.0 to 100.0. The default setting is 90, which means that a rising transition is considered to end when the signal rises to the voltage that is 90 percent of the rail-to-rail voltage.

This is one of eight variables, listed below, that specify the threshold measurement points of signal transitions. NanoTime uses these thresholds to calculate and report slew values and propagation delays in the presence of parasitics annotated with the **read_parasitics** command.

Variable Name	Default
rc_input_threshold_pct_fall	50.0
rc_input_threshold_pct_rise	50.0
rc_output_threshold_pct_fall	50.0
rc_output_threshold_pct_rise	50.0
rc_slew_lower_threshold_pct_fall	10.0
rc_slew_lower_threshold_pct_rise	10.0
rc_slew_upper_threshold_pct_fall	90.0

```
rc_slew_upper_threshold_pct_rise  90.0
```

With the default settings, NanoTime calculates each delay starting from the point where the input transition crosses the 50 percent threshold, and ending at the point where the output transition crosses the 50 percent threshold. To calculate slew for a rising transition, NanoTime considers the time from the 10 percent crossing point to the 90 percent crossing point; or for a falling transition, from the 90 percent crossing point to the 10 percent crossing point.

To specify these parameters for a particular net rather than globally, use the **set_measurement_threshold** command.

The following commands cause NanoTime to calculate delay from the point at which the input transition crosses the 50 percent threshold to the point at which the output transition crosses the 55 percent threshold (55 percent of the rail-to-rail voltage). Furthermore, to calculate transition times, NanoTime considers the time from the 20 percent to the 80 percent threshold crossing points for a rising transition, or from the 80 percent to the 20 percent threshold crossing points for a falling transition.

```
nt_shell> set rc_input_threshold_pct_fall 50
nt_shell> set rc_input_threshold_pct_rise 50
nt_shell> set rc_output_threshold_pct_fall 55
nt_shell> set rc_output_threshold_pct_rise 55
nt_shell> set rc_slew_lower_threshold_pct_fall 20
nt_shell> set rc_slew_lower_threshold_pct_rise 20
nt_shell> set rc_slew_upper_threshold_pct_fall 80
nt_shell> set rc_slew_upper_threshold_pct_rise 80
```

SEE ALSO

```
read_parasitics(2)
report_annotated_parasitics(2)
set_measurement_threshold(2)
rc_input_threshold_full_transition(3)
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_fall(3)
```

report_default_significant_digits

Sets the default number of significant digits displayed for floating-point values in reports.

TYPE

integer

DEFAULT

3

DESCRIPTION

This variable sets the default number of significant digits displayed for floating-point values in various reports. Allowed values are 0 to 13. Some report commands (for example, the **report_paths** command) have a **-significant_digits** option, which overrides the variable setting.

Not all reports are affected by this variable. Check the man pages for individual reporting commands to determine whether they support this feature.

SEE ALSO

`report_paths(2)`

report_paths_cell_hierarchy_level

Controls the display of cell (subcircuit) names in detailed path reports with respect to cell hierarchy level.

TYPE

integer

DEFAULT

0

DESCRIPTION

This variable controls the display of cell (subcircuit) names in detailed path reports. For each timing point in a detailed path or clock arrival report, a subcircuit name can be displayed.

A value of 0 for this variable (the default) displays the name of the subcircuit containing the cell that owns the pin of the timing point. A value of 1 displays the name of the subcircuit one level higher in the design hierarchy. A value of 2 displays the name of the subcircuit two levels higher, and so on.

A negative value disables the display of the subcircuit name with each timing point.

SEE ALSO

`report_paths(2)`
`report_clock_arrivals(2)`

report_paths_suppress_similar_paths_delay_ab

Specifies the maximum allowed absolute difference between the delays of different paths to be considered similar for reporting by the **report_paths** command with the **-suppress_similar_paths** option.

TYPE

float

DEFAULT

30

DESCRIPTION

The **-suppress_similar_paths** option of the **report_paths** command suppresses the display of multiple paths that are similar. Two paths are considered similar if they have similar timing and have one or more nets belonging to the same net group. Net groups are defined with the **create_net_group** command.

The following variables specify how closely the path delays, slacks, and stage delays must match in order to be considered similar:

```
report_paths_suppress_similar_paths_delay_absolute_tolerance
report_paths_suppress_similar_paths_delay_relative_tolerance
report_paths_suppress_similar_paths_slack_absolute_tolerance
report_paths_suppress_similar_paths_slack_relative_tolerance
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance
report_paths_suppress_similar_paths_stage_delay_relative_tolerance
```

The absolute settings specify the absolute allowable difference in picoseconds. The relative settings specify the allowable percentage difference.

The default of the **report_paths_suppress_similar_paths_delay_absolute_tolerance** variable is 30. This means that two paths are considered to have similar total delays if the delay values differ by no more than 30 picoseconds.

SEE ALSO

```
create_net_group(2)
report_paths(2)
report_paths_suppress_similar_paths_delay_relative_tolerance(3)
report_paths_suppress_similar_paths_slack_absolute_tolerance(3)
report_paths_suppress_similar_paths_slack_relative_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_relative_tolerance(3)
```

report_paths_suppress_similar_paths_delay_re

Specifies the maximum allowed relative difference between the delays of different paths to be considered similar for reporting by the **report_paths** command with the **-suppress_similar_paths** option.

TYPE

float

DEFAULT

2

DESCRIPTION

The **-suppress_similar_paths** option of the **report_paths** command suppresses the display of multiple paths that are similar. Two paths are considered similar if they have similar timing and have one or more nets belonging to the same net group. Net groups are defined with the **create_net_group** command.

The following variables specify how closely the path delays, slacks, and stage delays must match to be considered similar:

```
report_paths_suppress_similar_paths_delay_absolute_tolerance
report_paths_suppress_similar_paths_delay_relative_tolerance
report_paths_suppress_similar_paths_slack_absolute_tolerance
report_paths_suppress_similar_paths_slack_relative_tolerance
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance
report_paths_suppress_similar_paths_stage_delay_relative_tolerance
```

The absolute settings specify the absolute allowable difference in picoseconds. The relative settings specify the allowable percentage difference.

The default of the **report_paths_suppress_similar_paths_delay_relative_tolerance** variable is 2. This means that two paths are considered to have similar total delays if the delay values differ by no more than 2 percent.

SEE ALSO

```
create_net_group(2)
report_paths(2)
report_paths_suppress_similar_paths_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_slack_absolute_tolerance(3)
report_paths_suppress_similar_paths_slack_relative_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_relative_tolerance(3)
```

report_paths_suppress_similar_paths_slack_ab

Specifies the maximum allowed absolute difference between the slacks of different paths to be considered similar for reporting by the **report_paths** command with the **-suppress_similar_paths** option.

TYPE

float

DEFAULT

30

DESCRIPTION

The **-suppress_similar_paths** option of the **report_paths** command suppresses the display of multiple paths that are similar. Two paths are considered similar if they have similar timing and have one or more nets belonging to the same net group. Net groups are defined with the **create_net_group** command.

The following variables specify how closely the path delays, slacks, and stage delays must match to be considered similar:

```
report_paths_suppress_similar_paths_delay_absolute_tolerance
report_paths_suppress_similar_paths_delay_relative_tolerance
report_paths_suppress_similar_paths_slack_absolute_tolerance
report_paths_suppress_similar_paths_slack_relative_tolerance
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance
report_paths_suppress_similar_paths_stage_delay_relative_tolerance
```

The absolute settings specify the absolute allowable difference in picoseconds. The relative settings specify the allowable percentage difference.

The default of the **report_paths_suppress_similar_paths_slack_absolute_tolerance** variable is 30. This means that two paths are considered to have similar slacks if the slack values differ by no more than 30 picoseconds.

SEE ALSO

```
create_net_group(2)
report_paths(2)
report_paths_suppress_similar_paths_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_delay_relative_tolerance(3)
report_paths_suppress_similar_paths_slack_relative_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_relative_tolerance(3)
```

report_paths_suppress_similar_paths_slack_rel

Specifies the maximum allowed relative difference between the slacks of different paths to be considered similar for reporting by the **report_paths** command with the **-suppress_similar_paths** option.

TYPE

float

DEFAULT

2

DESCRIPTION

The **-suppress_similar_paths** option of the **report_paths** command suppresses the display of multiple paths that are similar. Two paths are considered similar if they have similar timing and have one or more nets belonging to the same net group. Net groups are defined with the **create_net_group** command.

The following variables specify how closely the path delays, slacks, and stage delays must match to be considered similar:

```
report_paths_suppress_similar_paths_delay_absolute_tolerance
report_paths_suppress_similar_paths_delay_relative_tolerance
report_paths_suppress_similar_paths_slack_absolute_tolerance
report_paths_suppress_similar_paths_slack_relative_tolerance
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance
report_paths_suppress_similar_paths_stage_delay_relative_tolerance
```

The absolute settings specify the absolute allowable difference in picoseconds. The relative settings specify the allowable percentage difference.

The default of the **report_paths_suppress_similar_paths_slack_relative_tolerance** variable is 2. This means that two paths are considered to have similar slacks if the slack values differ by no more than 2 percent.

SEE ALSO

create_net_group(2)
report_paths(2)
report_paths_suppress_similar_paths_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_delay_relative_tolerance(3)
report_paths_suppress_similar_paths_slack_absolute_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_relative_tolerance(3)

report_paths_suppress_similar_paths_stage_de

Specifies the maximum allowed absolute difference between the stage delays of different paths to be considered similar for reporting by the **report_paths** command with the **-suppress_similar_paths** option.

TYPE

float

DEFAULT

10

DESCRIPTION

The **-suppress_similar_paths** option of the **report_paths** command suppresses the display of multiple paths that are similar. Two paths are considered similar if they have similar timing and have one or more nets belonging to the same net group. Net groups are defined with the **create_net_group** command.

The following variables specify how closely the path delays, slacks, and stage delays must match to be considered similar:

```
report_paths_suppress_similar_paths_delay_absolute_tolerance
report_paths_suppress_similar_paths_delay_relative_tolerance
report_paths_suppress_similar_paths_slack_absolute_tolerance
report_paths_suppress_similar_paths_slack_relative_tolerance
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance
report_paths_suppress_similar_paths_stage_delay_relative_tolerance
```

The absolute settings specify the absolute allowable difference in picoseconds. The relative settings specify the allowable percentage difference.

The default of the **report_paths_suppress_similar_paths_stage_delay_absolute_tolerance** variable is 10. This means that two paths are considered to have similar stage delays if the stage delay

values differ by no more than 10 picoseconds.

SEE ALSO

create_net_group(2)
report_paths(2)
report_paths_suppress_similar_paths_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_delay_relative_tolerance(3)
report_paths_suppress_similar_paths_slack_absolute_tolerance(3)
report_paths_suppress_similar_paths_slack_relative_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_relative_tolerance(3)

report_paths_suppress_similar_paths_stage_de

Specifies the maximum allowed relative difference between the stage delays of different paths to be considered similar for reporting by the **report_paths** command with the **-suppress_similar_paths** option.

TYPE

float

DEFAULT

2

DESCRIPTION

The **-suppress_similar_paths** option of the **report_paths** command suppresses the display of multiple paths that are similar. Two paths are considered similar if they have similar timing and have one or more nets belonging to the same net group. Net groups are defined with the **create_net_group** command.

The following variables specify how closely the path delays, slacks, and stage delays must match to be considered similar:

```
report_paths_suppress_similar_paths_delay_absolute_tolerance
report_paths_suppress_similar_paths_delay_relative_tolerance
report_paths_suppress_similar_paths_slack_absolute_tolerance
report_paths_suppress_similar_paths_slack_relative_tolerance
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance
report_paths_suppress_similar_paths_stage_delay_relative_tolerance
```

The absolute settings specify the absolute allowable difference in picoseconds. The relative settings specify the allowable percentage difference.

The default of the **report_paths_suppress_similar_paths_stage_delay_relative_tolerance** variable is 2. This means that two paths are considered to have similar stage delays if the stage delay values differ

by no more than 2 percent.

SEE ALSO

```
create_net_group(2)
report_paths(2)
report_paths_suppress_similar_paths_delay_absolute_tolerance(3)
report_paths_suppress_similar_paths_delay_relative_tolerance(3)
report_paths_suppress_similar_paths_slack_absolute_tolerance(3)
report_paths_suppress_similar_paths_slack_relative_tolerance(3)
report_paths_suppress_similar_paths_stage_delay_absolute_tolerance(3)
```

search_path

Specifies the directories in which NanoTime searches for files.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies a list of directories that NanoTime searches to find design files. It affects file searching by the following commands: **link_design**, **read_device_parameters**, **read_library**, **read_parasitics**, **read_pattern**, **read_spice_model**, **read_techfile**, and **register_netlist**. Each of these commands can read in one or more specified files. NanoTime searches for the files in the directories listed in the **search_path** variable, in the order listed.

By default, the **source** command is not affected by the **search_path** setting. To cause the **source** command to search for scripts in the directories specified by the **search_path** variable, set the **sh_source_uses_search_path** variable to **true**.

SEE ALSO

```
link_design(2)
read_device_parameters(2)
read_library(2)
read_parasitics(2)
```

```
read_pattern(2)
read_spice_model(2)
read_techfile(2)
register_netlist(2)
source(2)
library_path(3)
link_path(3)
sh_source_uses_search_path(3)
```

sh_allow_tcl_with_set_app_var

Allows the **set_app_var** and **get_app_var** commands to work with application variables.

TYPE

string

DEFAULT

application specific

DESCRIPTION

Normally the **get_app_var** and **set_app_var** commands only work for variables that have been registered as application variables. Setting this variable to **true** allows these commands to set a Tcl global variable instead.

These commands issue a CMD-104 error message for the Tcl global variable, unless the variable name is included in the list specified by the **sh_allow_tcl_with_set_app_var_no_message_list** variable.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var_no_message_list(2)
```

sh_allow_tcl_with_set_app_var_no_message_l

Suppresses CMD-104 messages for variables in this list.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable is consulted before printing the CMD-104 error message, if the **sh_allow_tcl_with_set_app_var** variable is set to **true**. All variables in this Tcl list receive no message.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var(2)
```

sh_arch

Indicates the system architecture of your machine.

TYPE

string

DEFAULT

platform-dependent

DESCRIPTION

The **sh_arch** variable is set by the application to indicate the system architecture of your machine. Examples of machines being used are sparcOS5, amd64, and so on. This variable is read-only.

sh_command_abbrev_mode

Sets the command abbreviation mode for interactive convenience.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable sets the command abbreviation mode as an interactive convenience. Script files should not use any command or option abbreviation, because these files are then susceptible to command changes in subsequent versions of the application.

Although the default value is **Anywhere**, it is recommended that the site startup file for the application set this variable to **Command-Line-Only**. It is also possible to set the value to **None**, which disables abbreviations altogether.

To determine the current value of this variable, use the **get_app_var sh_command_abbrev_mode** command.

SEE ALSO

sh_command_abbrev_options(3)
get_app_var(2)
set_app_var(2)

sh_command_abbrev_options

Turns off abbreviation of command dash option names when false.

TYPE

boolean

DEFAULT

application specific

DESCRIPTION

When command abbreviation is currently off (see sh_command_abbrev_mode) then setting this variable to false will also not allow abbreviation of command dash options. This variable also impacts abbreviation of the values specified to command options that expect values to be one of an allowed list of values.

This variable exists to be backward compatible with previous tool releases which always allowed abbreviation of command dash options and option values regardless of the command abbreviation mode.

It is recommended to set the value of this variable to false.

To determine the current value of this variable, use the **get_app_var sh_command_abbrev_options** command.

SEE ALSO

sh_command_abbrev_mode(3)
get_app_var(2)
set_app_var(2)

sh_command_log_file

Specifies the name of the file to which the application logs the commands you executed during the session.

TYPE

string

DEFAULT

empty string

DESCRIPTION

This variable specifies the name of the file to which the application logs the commands you run during the session. By default, the variable is set to an empty string, indicating that the application's default command log file name is to be be used. If a file named by the default command log file name cannot be opened (for example, if it has been set to read only access), then no logging occurs during the session.

This variable can be set at any time. If the value for the log file name is invalid, the variable is not set, and the current log file persists.

To determine the current value of this variable, use the **get_app_var sh_command_log_file** command.

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_continue_on_error

Allows processing to continue when errors occur during script execution with the **source** command.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

This variable is deprecated. It is recommended to use the **-continue_on_error** option to the **source** command instead of this variable because that option only applies to a single script, and not the entire application session.

When set to **true**, the **sh_continue_on_error** variable allows processing to continue when errors occur. Under normal circumstances, when executing a script with the **source** command, Tcl errors (syntax and semantic) cause the execution of the script to terminate.

When **sh_continue_on_error** is set to **false**, script execution can also terminate due to new error and warning messages based on the value of the **sh_script_stop_severity** variable.

To determine the current value of the **sh_continue_on_error** variable, use the **get_app_var sh_continue_on_error** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_script_stop_severity(3)
```

sh_deprecated_is_error

Raise a Tcl error when a deprecated command is executed.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set this variable causes a Tcl error to be raised when an deprecated command is executed. Normally only a warning message is issued.

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_dev_null

Indicates the current null device.

TYPE

string

DEFAULT

platform dependent

DESCRIPTION

This variable is set by the application to indicate the current null device. For example, on UNIX machines, the variable is set to **/dev/null**. This variable is read-only.

SEE ALSO

`get_app_var(2)`

sh_enable_line_editing

Enables command line editing capabilities. This variable is for use in Tcl mode only.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Enables advanced unix-like shell capabilities. This variable is for use in Tcl mode only. The variable must be set in a setup file (a file whose name ends in .synopsys_nt.setup) to take effect.

Key Bindings

The **sh_list_key_bindings** command displays current key bindings and the edit mode. To change the edit mode, use the **sh_line_editing_mode** variable on the shell.

Command Completion

The editor can complete commands, options, variables and files, given a unique abbreviation. Type part of a word and press the Tab key to see the complete command, variable or file name. For command options, type a command name followed by the dash character (-) and press the Tab key to see the list of options for that command.

If no match is found, the terminal bell rings. If the word is already complete, a space is added to the end (if it isn't already there) to speed typing and provide a visual indicator of successful completion. Completed text pushes the rest of the line to the right. If there are multiple matches, all of the matching commands, options, files or variables are listed.

Completion works in a context-sensitive way, as follows.

```
The first token of a command line          :  completes commands

Token that begins with "-" after a command :  completes command arguments

After >, | or a sh_* command               :  completes file names

After a set, unset or printvar command     :  completes variables

After the $ symbol                         :  completes variables

After the help command                     :  completes command

After the man command                      :  completes commands or variables

Any token which is not the first token
and doesn't match any of the other rules   :  completes file names
```

SEE ALSO

```
sh_line_editing_mode(3)
sh_list_key_bindings(2)
set_cle_options(2)
```

sh_enable_message_context_with_delimiter

Enables adding message context information to formal messages (information, warnings, and errors).

TYPE

string

DEFAULT

""

DESCRIPTION

If you set this variable to a delimiter string, the delimiter string and message context information are added to a formal message such that the new message will be in the following form:

```
<original message> <delimiter string> <message context information>
```

The message context information contains the following, if available:

- The net name: The name of the most significant net related to the message.
- The instance name: The full instance name of the most related subcircuit.
- The subcircuit name: The subcircuit (not a transistor wrapper) name of the instance.
- The transition direction: The path tracing edge direction of the net.

The syntax for added context information is as follows:

```
[net=<net_name>, inst=<inst_name>, subckt=<subckt_name>, edge=<rising|falling>]
```

Any string can be used for the delimiter, but it is best to choose a string that is unique and does not confuse the meaning of the reported message.

For example, set the delimiter to the string '@&#' as follows:

```
nt_shell> set sh_enable_message_context_with_delimiter @&#
```

An example of a warning message with context information is as follows:

```
Warning: Failed to get delay from pin XI1/MM0/G (net clk) rising to XI2/MM0/G
(net z) falling, unable to trace further. (DELC-003) @&# [net=clk, inst=XI1,
subckt=INV, edge=rising] [net=z, inst=XI2, subckt=INV, edge=falling]
```

Messages that do not have additional information to report are displayed with the delimiter string and empty context, as in the following example:

```
Warning: Specified default supply voltage of 1.300V is outside the range
(1.200V, 1.200V) of supply voltages found in the design. (OPCD-008) @&#
```

sh_enable_page_mode

Displays long reports one page at a time (similar to the UNIX **more** command.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

This variable displays long reports one page at a time (similar to the UNIX **more** command), when set to **true**. Consult the man pages for the commands that generate reports to see if they are affected by this variable.

To determine the current value of this variable, use the **get_app_var sh_enable_page_mode** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_enable_stdout_redirect

Allows the redirect command to capture output to the Tcl stdout channel.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set to **true**, this variable allows the redirect command to capture output sent to the Tcl stdout channel. By default, the Tcl **puts** command sends its output to the stdout channel.

SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

sh_help_shows_group_overview

Changes the behavior of the "help" command.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable changes the behavior of the **help** command when no arguments are specified to help. Normally when no arguments are specified an informational message with a list of available command groups is displayed.

When this variable is set to false the command groups and the commands in each group is printed instead. This variable exists for backward compatibility.

SEE ALSO

`help(2)`
`set_app_var(2)`

sh_line_editing_mode

Enables vi or emacs editing mode. This variable is for use in Tcl mode only.

TYPE

string

DEFAULT

emacs

DESCRIPTION

The **sh_line_editing_mode** variable is used to set the command line editor mode. Valid values are **emacs** and **vi**.

Use the **sh_list_key_bindings** command to display the current key bindings and edit mode.

This variable is for use in Tcl mode only. It can be set on the shell or inside a setup file (a file whose name ends in .synopsys_nt.setup) to take effect.

SEE ALSO

```
sh_enable_line_editing(3)
sh_list_key_bindings(2)
set_cle_options(2)
```

sh_new_variable_message

Controls a debugging feature for tracing the creation of new variables.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

The **sh_new_variable_message** variable controls a debugging feature for tracing the creation of new variables. Its primary debugging purpose is to catch the misspelling of an application-owned global variable. When set to **true**, an informational message (CMD-041) is displayed when a variable is defined for the first time at the command line. When set to **false**, no message is displayed.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var command man page for details.**

Other variables, in combination with **sh_new_variable_message**, enable tracing of new variables in scripts and Tcl procedures.

Warning: This feature has a significant negative impact on CPU performance when used with scripts and Tcl procedures. This feature should be used only when developing scripts or in interactive use. When you turn on the feature for scripts or Tcl procedures, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message_in_proc(3)
sh_new_variable_message_in_script(3)
```

sh_new_variable_message_in_proc

Controls a debugging feature for tracing the creation of new variables in a Tcl procedure.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_proc** variable controls a debugging feature for tracing the creation of new variables in a Tcl procedure. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. Please see the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. Enabling the feature simply enables the **print_proc_new_vars** command. In order to trace the creation of variables in a procedure, this command must be inserted into the procedure, typically as the last statement. When all of these steps have been taken, an informational message (CMD-041) is generated for new variables defined within the procedure, up to the point that the **print_proc_new_vars** commands is executed.

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_proc** command.

SEE ALSO

```
get_app_var(2)
print_proc_new_vars(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_script(3)
```

sh_new_variable_message_in_script

Controls a debugging feature for tracing the creation of new variables within a sourced script.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_script** variable controls a debugging feature for tracing the creation of new variables within a sourced script. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. In that case, an informational message (CMD-041) is displayed when a variable is defined for the first time. When **sh_new_variable_message_in_script** is set to **false** (the default), no message is displayed at the time that the variable is created. When the **source** command completes, however, you see messages for any new variables that were created in the script. This is because the state of the variables is sampled before and after the **source** command. It is not because of inter-command sampling within the script. So, this is actually a more efficient method to see if new variables were created in the script.

For example, given the following script a.tcl:

```
echo "Entering script"
set a 23
echo a = $a
```

```
set b 24
echo b = $b
echo "Exiting script"
```

When **sh_new_variable_message_in_script** is **false** (the default), you see the following when you source the script:

```
prompt> source a.tcl
Entering script
a = 23
b = 24
Exiting script
Information: Defining new variable 'a'. (CMD-041)
Information: Defining new variable 'b'. (CMD-041)
prompt>
```

Alternatively, when **sh_new_variable_message_in_script** is **true**, at much greater cost, you see the following when you source the script:

```
prompt> set sh_new_variable_message_in_script true
Warning: Enabled new variable message tracing -
         Tcl scripting optimization disabled. (CMD-042)
true
prompt> source a.tcl
Entering script
Information: Defining new variable 'a'. (CMD-041)
a = 23
Information: Defining new variable 'b'. (CMD-041)
b = 24
Exiting script
prompt>
```

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_script** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_proc(3)
```

sh_obsolete_is_error

Raise a Tcl error when an obsolete command is executed.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set this variable causes a Tcl error to be raised when an obsolete command is executed. Normally only a warning message is issued.

Obsolete commands have no effect.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_product_version

Indicates the version of the application currently running.

TYPE

string

DESCRIPTION

This variable is set to the version of the application currently running. The variable is read only.

To determine the current value of this variable, use the **get_app_var sh_product_version** command.

SEE ALSO

`get_app_var(2)`

sh_script_stop_severity

Indicates the error message severity level that would cause a script to stop running before it completes.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is run with the **source** command, there are several ways to get it to stop running before it completes. One is to use the **sh_script_stop_severity** variable. This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a script to stop.
- When set to **W**, the generation of one or more warning or error messages causes a script to stop.
- When set to **none**, the generation messages does not cause the script to stop.

Note that **sh_script_stop_severity** is ignored if **sh_continue_on_error** is set to **true**.

To determine the current value of this variable, use the **get_app_var sh_script_stop_severity** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
```

sh_source_emits_line_numbers

Indicates the error message severity level that causes an informational message to be issued, listing the script name and line number where that message occurred.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is executed with the **source** command, error and warning messages can be emitted from any command within the script. Using the **sh_source_emits_line_numbers** variable, you can help isolate where errors and warnings are occurring.

This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a CMD-082 informational message to be issued when the command completes, giving the name of the script and the line number of the command.
- When set to **W**, the generation of one or more warning or error messages causes a the CMD-082 message.

The setting of **sh_script_stop_severity** affects the output of the CMD-082 message. If the setting of **sh_script_stop_severity** causes a CMD-081 message, then it takes precedence over CMD-082.

To determine the current value of this variable, use the **get_app_var sh_source_emits_line_numbers** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
sh_script_stop_severity(3)
CMD-081(n)
CMD-082(n)
```

sh_source_logging

Indicates if individual commands from a sourced script should be logged to the command log file.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When you source a script, the **source** command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment. You can disable this logging by setting **sh_source_logging** to **false**.

To determine the current value of this variable, use the **get_app_var sh_source_logging** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
```

sh_source_uses_search_path

Indicates if the **source** command uses the **search_path** variable to search for files.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When this variable is set to \ftrue the **source** command uses the **search_path** variable to search for files. When set to **false**, the **source** command considers its file argument literally.

To determine the current value of this variable, use the **get_app_var sh_source_uses_search_path** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
search_path(3)
```

sh_tcllib_app_dirname

Indicates the name of a directory where application-specific Tcl files are found.

TYPE

string

DESCRIPTION

The **sh_tcllib_app_dirname** variable is set by the application to indicate the directory where application-specific Tcl files and packages are found. This is a read-only variable.

SEE ALSO

`get_app_var(2)`

sh_user_man_path

Indicates a directory root where you can store man pages for display with the **man** command.

TYPE

list

DEFAULT

empty list

DESCRIPTION

The **sh_user_man_path** variable is used to indicate a directory root where you can store man pages for display with the **man** command. The directory structure must start with a directory named *man*. Below *man* are directories named *cat1*, *cat2*, *cat3*, and so on. The **man** command will look in these directories for files named *file.1*, *file.2*, and *file.3*, respectively. These are pre-formatted files. It is up to you to format the files. The **man** command effectively just types the file.

These man pages could be for your Tcl procedures. The combination of defining help for your Tcl procedures with the **define_proc_attributes** command, and keeping a manual page for the same procedures allows you to fully document your application extensions.

The **man** command will look in **sh_user_man_path** after first looking in application-defined paths. The user-defined paths are consulted only if no matches are found in the application-defined paths.

To determine the current value of this variable, use the **get_app_var sh_user_man_path** command.

SEE ALSO

```
define_proc_attributes(2)
get_app_var(2)
man(2)
set_app_var(2)
```

si_aggressive_filtering_per_aggr

Allows a single filtering criterion to cause an aggressor net to be filtered instead of requiring a combination of criteria.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime uses a combination of criteria during the parasitic filtering phase for aggressor nets impacting a specific victim net. By default, for an aggressor net to be filtered, all of the thresholds specified by the **si_filter_per_aggr_xcap**, **si_filter_per_aggr_xcap_to_gcap_ratio** and **si_filter_per_aggr_to_average_aggr_xcap_ratio** variables must be met. If the **si_aggressive_filtering_per_aggr** variable is set to **true**, only one of the three criteria must be met for the aggressor net to be filtered.

SEE ALSO

si_enable_analysis(3)
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)
si_filter_per_aggr_xcap(3)
si_filter_per_aggr_xcap_to_gcap_ratio(3)
si_filter_total_aggr_xcap_to_gcap_ratio(3)
si_filter_total_aggr_xcap(3)
si_aggressive_filtering_total(3)

si_aggressive_filtering_total

Allows a single filtering criterion to cause a victim net to be filtered instead of requiring a combination of criteria.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime uses a combination of criteria during the parasitic filtering phase for victim nets. By default, for a victim net to be filtered, both of the thresholds specified by the **si_filter_total_aggr_xcap** and **si_filter_total_aggr_xcap_to_gcap_ratio** variables must be met. If the **si_aggressive_filtering_total** variable is set to **true**, only one of the two criteria must be met for the victim net to be filtered.

SEE ALSO

si_enable_analysis(3)
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)
si_filter_per_aggr_xcap(3)
si_filter_per_aggr_xcap_to_gcap_ratio(3)
si_filter_total_aggr_xcap_to_gcap_ratio(3)
si_filter_total_aggr_xcap(3)
si_aggressive_filtering_per_aggr(3)

si_aggressor_transition_default_fall

Specifies the default fall transition time used for a crosstalk aggressor which did not have a valid timing window and transition time computed during path tracing.

TYPE

float

DEFAULT

0.05

DESCRIPTION

During crosstalk delay and noise analysis the switching window and transition time of each aggressor is used to compute its impact on a victim.

If the input stimulus for the original analysis does not exercise every path through the design during path tracing then some aggressor nets might not have a valid timing window and transition time.

The values of the **si_aggressor_transition_default_fall** and **si_aggressor_transition_default_rise** variables are used to model the impact from these aggressors. To effectively ignore aggressors without timing windows and transition times, these variables should each be set to a large value.

The value is in user time units.

SEE ALSO

`si_enable_analysis(3)`

```
si_enable_noise_analysis(3)
si_aggressor_transition_default_rise(3)
report_crosstalk_delay_sources(2)
report_noise_violation_sources(2)
```

si_aggressor_transition_default_rise

Specifies the default rise transition time used for a crosstalk aggressor which did not have a valid timing window and transition time computed during path tracing.

TYPE

float

DEFAULT

0.05

DESCRIPTION

During crosstalk delay and noise analysis the switching window and transition time of each aggressor is used to compute its impact on a victim.

If the input stimulus for the original analysis does not exercise every path through the design during path tracing then some aggressor nets might not have a valid timing window and transition time.

The values of the **si_aggressor_transition_default_fall** and **si_aggressor_transition_default_rise** variables are used to model the impact from these aggressors. To effectively ignore aggressors without timing windows and transition times, these variables should each be set to a large value.

The value is in user time units.

SEE ALSO

`si_enable_analysis(3)`

```
si_enable_noise_analysis(3)
si_aggressor_transition_default_fall(3)
report_crosstalk_delay_sources(2)
report_noise_violation_sources(2)
```

si_enable_aggressor_logic_pessimism_reductio

Enables or disables the consideration of pessimism reduction due to logic constraints among aggressors during signal integrity (crosstalk) delay and noise analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime signal integrity (crosstalk) delay and noise analysis both consider logic constraints among aggressors in order to reduce pessimism. By default, this variable is set to **false**.

To enable crosstalk delay analysis, you must have a NanoTime Ultra license and set the **si_enable_analysis** variable to **true**. To enable crosstalk noise analysis, you must have a NanoTime Ultra license and set the **si_enable_noise_analysis** variable to **true**.

SEE ALSO

si_enable_analysis(3)
si_enable_noise_analysis(3)

si_enable_analysis

Enables or disables signal integrity (crosstalk) delay analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime performs crosstalk-aware delay calculation during path tracing. By default, the variable is set to **false** and crosstalk delay analysis is disabled.

This variable must be set before executing the **link_design** command.

To retain the cross-coupling capacitors when you read in parasitics from a DSPF or SPEF file, use the **-keep_capacitive_coupling** option with the **read_parasitics** command. If you are using a SPICE netlist with embedded parasitics, use the **-keep_capacitive_coupling** option with the **link_design** command.

To enable crosstalk delay analysis, you must have a NanoTime Ultra license.

SEE ALSO

```
link_design(2)
read_parasitics(2)
```

si_enable_first_iteration_pessimism_reduction

Enables the consideration of logic constraints between victims and aggressors during the first iteration of NanoTime signal integrity (crosstalk) delay analysis.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default), the first iteration of NanoTime signal integrity analysis considers logic constraints between victims and aggressors to reduce pessimism. In addition, if the **si_enable_aggressor_logic_pessimism_reduction** variable is set to **true**, logic constraints among aggressors are also considered during the first iteration to further reduce pessimism.

In cases where NanoTime signal integrity analysis requires multiple iterations, first iteration pessimism reduction might cause increased total runtime. You can disable the feature by setting the variable to **false**.

To enable crosstalk delay analysis, you must have a NanoTime Ultra license and set the **si_enable_analysis** variable to **true**.

SEE ALSO

```
si_enable_analysis(3)
si_enable_aggressor_logic_pessimism_reduction(3)
```

si_enable_noise_aggressor_windows_pessimisn

Enables or disables the consideration of pessimism reduction due to aggressor timing windows during signal integrity (crosstalk) noise analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime signal integrity (crosstalk) noise analysis considers aggressor timing windows in order to reduce pessimism. By default, this variable is set to **false**.

If the **si_enable_aggressor_logic_pessimism_reduction** variable is set to **true** to enable consideration of aggressor-to-aggressor logic constraints during noise analysis, NanoTime also considers aggressor timing windows in the analysis, regardless of the value of the **si_enable_noise_aggressor_windows_pessimism_reduction** variable.

To enable crosstalk noise analysis, you must have a NanoTime Ultra license and set the **si_enable_noise_analysis** variable to **true**.

SEE ALSO

```
si_enable_noise_analysis(3)
si_enable_aggressor_logic_pessimism_reduction(3)
```

si_enable_noise_analysis

Enables or disables signal integrity (crosstalk) noise analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime performs crosstalk noise analysis. By default, the variable is set to **false** and noise analysis is disabled.

This variable must be set before executing the **link_design** command.

To retain the cross-coupling capacitors when you read in parasitics from a DSPF or SPEF file, use the **-keep_capacitive_coupling** option with the **read_parasitics** command. If you are using a SPICE netlist with embedded parasitics, use the **-keep_capacitive_coupling** option with the **link_design** command.

To enable crosstalk noise analysis, you must have a NanoTime Ultra license.

SEE ALSO

```
link_design(2)
read_parasitics(2)
update_noise(2)
```

si_enable_noise_fanout_analysis

Enables or disables the ability to compute the impact of noise from a victim net and how it propagates forward through each channel-connected region it drives.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime performs crosstalk-aware noise calculations of fanout noise values with the **update_noise** command. The impact of noise on each victim net is determined for each fanout driven by the victim net. By default, this variable is set to **false** and fanout noise is not computed.

The **si_fanout_noise_threshold_above_low** and **si_fanout_noise_threshold_below_high** variables control whether propagated noise is computed.

To retain the cross-coupling capacitors when you read in parasitics from a DSPF or SPEF file, use the **-keep_capacitive_coupling** option with the **read_parasitics** command. If you are using a SPICE netlist with embedded parasitics, use the **-keep_capacitive_coupling** option with the **link_design** command.

This variable must be set before executing the **link_design** command.

To enable crosstalk noise analysis, you must have a NanoTime Ultra license and set the **si_enable_noise_analysis** variable to **true**.

SEE ALSO

```
read_parasitics(2)
report_fanout_noise(2)
si_enable_noise_analysis(3)
set_fanout_noise_threshold(2)
si_fanout_noise_threshold_above_low(3)
si_fanout_noise_threshold_below_high(3)
```

si_exclusion_cap_factor_max

Specifies the default coupling capacitance factor to be used during maximum delay analysis for nets that are excluded from signal integrity (crosstalk) analysis.

TYPE

float

DEFAULT

1.0

DESCRIPTION

Specifies the default coupling capacitance factor to be used when nets are excluded from maximum delay crosstalk analysis. The valid range is 0.0 to 2.0, both numbers inclusive. NanoTime splits the coupling capacitances of excluded nets to ground and multiplies them by this factor.

SEE ALSO

```
si_enable_analysis(3)
set_si_delay_analysis(2)
si_exclusion_cap_factor_min(3)
```

si_exclusion_cap_factor_min

Specifies the default coupling capacitance factor to be used during minimum delay analysis for nets that are excluded from signal integrity (crosstalk) analysis.

TYPE

float

DEFAULT

1.0

DESCRIPTION

Specifies the default coupling capacitance factor to be used when nets are excluded from minimum delay crosstalk analysis. The valid range is 0.0 to 2.0, both numbers inclusive. NanoTime splits the coupling capacitances of excluded nets to ground and multiplies them by this factor.

SEE ALSO

```
si_enable_analysis(3)
set_si_delay_analysis(2)
si_exclusion_cap_factor_max(3)
```

si_fanout_noise_margin_above_low

Specifies the amount of voltage allowed for fanout noise values above the ground rail.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of noise voltage above the ground rail allowed before being considered a violation. It is specified in volts.

By default, the margin is set to zero indicating that no noise is allowed. Specify a positive value for a less restrictive check.

SEE ALSO

```
update_noise(2)
report_noise(2)
report_fanout_noise(2)
si_fanout_noise_margin_below_high(3)
```

si_fanout_noise_margin_below_high

Specifies the amount of voltage allowed for fanout noise values below the supply rail.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of noise below the supply rail allowed before being considered a violation. It is specified in volts.

By default, the margin is set to zero indicating that no noise is allowed. Specify a positive value for a less restrictive check.

SEE ALSO

```
update_noise(2)
report_noise(2)
report_fanout_noise(2)
si_fanout_noise_margin_above_low(3)
```

si_fanout_noise_threshold_above_low

Specifies the threshold voltage that determines if the noise peak above the ground rail on a victim net should be propagated forward to compute its impact on subsequent fanout nets.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the threshold above the ground voltage that must be exceeded before a propagated noise calculation is made. If the noise peak is exceeded then further simulations take place to compute the impact of the noise peak on each fanout net.

The default is 0, which means that every noise peak will be propagated forward.

This is one of two variables that specify the thresholds used to determine if fanout noise values are computed.

Variable Name	Default
si_fanout_noise_threshold_above_low	0.0
si_fanout_noise_threshold_below_high	0.0

To specify these parameters for a particular net rather than globally, use the **set_fanout_noise_threshold** command.

The following commands cause NanoTime to propagate noise values which are .05 volts above the ground rail or .07 volts below the supply rail.

```
nt_shell> set si_fanout_noise_threshold_above_low  .05
nt_shell> set si_fanout_noise_threshold_below_high .07
```

SEE ALSO

update_noise(2)
report_fanout_noise(2)
set_fanout_noise_threshold(2)
si_fanout_noise_threshold_below_high(3)

si_fanout_noise_threshold_below_high

Specifies the threshold voltage that determines if the noise peak below the supply rail on a victim net should be propagated forward to compute its impact on subsequent fanout nets.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the threshold below the supply voltage that must be exceeded before a propagated noise calculation is made. If the noise peak is exceeded then further simulations take place to compute the impact of the noise peak on each fanout net.

The default is 0, which means that every noise peak will be propagated forward.

This is one of two variables that specify the thresholds used to determine if fanout noise values are computed.

Variable Name	Default
si_fanout_noise_threshold_above_low	0.0
si_fanout_noise_threshold_below_high	0.0

To specify these parameters for a particular net rather than globally, use the **set_fanout_noise_threshold** command.

The following commands cause NanoTime to propagate noise values which are .05 volts above the ground rail or .07 volts below the supply rail.

```
nt_shell> set si_fanout_noise_threshold_above_low  .05
nt_shell> set si_fanout_noise_threshold_below_high .07
```

SEE ALSO

update_noise(2)
report_fanout_noise(2)
set_fanout_noise_threshold(2)
si_fanout_noise_threshold_above_low(3)

si_filter_per_aggr_to_average_aggr_xcap_ratio

Specifies the minimum value of the ratio of the total cross-coupled capacitance between an aggressor net and a victim net to the average cross-coupled capacitance between the victim net and all of its aggressor nets, below which the aggressor net can be filtered during parasitic filtering.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_filter_per_aggr_to_average_aggr_xcap_ratio** variable specifies the threshold of the ratio of the total cross-coupled capacitance between an aggressor net and a victim net to the average cross-coupled capacitance between the victim net and all of its aggressor nets.

NanoTime signal integrity analysis uses three variables (including this one) during the second stage of parasitic filtering to determine whether an aggressor net can be filtered for a particular victim net:

```
si_filter_per_aggr_xcap
si_filter_per_aggr_xcap_to_gcap_ratio
si_filter_per_aggr_to_average_aggr_xcap_ratio
```

NanoTime evaluates an aggressor net for possible filtering based on the conditions described in the "Filtering Criteria" section. A filtered aggressor net can still be considered as an aggressor net to other victim nets. Note that a coupling capacitor can be filtered at one end and not at the other. You disable this type of filtering by setting the **si_filter_per_aggr_xcap** variable to zero.

If the **si_aggressive_filtering_per_aggr** is set to **true**, satisfying any one of the criteria allows this net to be filtered. If the variable is **false** (the default), all conditions must be met.

Filtering Criteria

Define the ground capacitance of each subnode of a victim V as

```
Cg(V)[i]
where i = from 1 to Ng(V)
```

Then, the total ground capacitance of V is

```
Cg(V) = Cg(V)[1] + Cg(V)[2] + ... + Cg(V)[Ng(V)]
```

Further, define the coupling capacitors between a subnode of victim V and a subnode of its aggressor j Aj as

```
Cc(V, Aj)[k]
where j = from 1 to Na(V)and k = from 1 to Nc(V, Aj)
```

Then, the total coupling capacitance between V and Aj is

```
Cc(V, Aj) = Cc(V, Aj)[1] + Cc(V, Aj)[2] + ... + Cc(V, Aj)[Nc(V, Aj)]
```

and the total coupling capacitance of V to all its aggressors is

```
Cc(V) = Cc(V, A1) + Cc(V, A2) + ... + Cc(V, AK)
where K is Na(V)
```

The filtering conditions are as follows:

- `Cc(V, Aj) is less than the value of the` **`si_filter_per_aggr_xcap`** `variable.`
- `Cc(V, Aj)/(Cg(V)+Cc(V)) is less than the value of the` **`si_filter_per_aggr_xcap_to_gcap_ratio`** `variable.`
- `Cc(V, Aj)*Na(V)/Cc(V) is less than the value of the` **`si_filter_per_aggr_to_average_aggr_xcap_ratio`** `variable.`

SEE ALSO

```
si_enable_analysis(3)
si_filter_per_aggr_xcap(3)
si_filter_per_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_per_aggr(3)
si_filter_total_aggr_xcap(3)
si_filter_total_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_total(3)
```

si_filter_per_aggr_xcap

Specifies the minimum value of the total cross-coupled capacitance between an aggressor net and a victim net, below which the aggressor net can be filtered during parasitic filtering.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_filter_per_aggr_xcap** variable specifies the threshold, in library units, of the total cross-coupled capacitance between an aggressor net and a victim net.

NanoTime signal integrity analysis uses three variables (including this one) during the second stage of parasitic filtering to determine whether an aggressor net can be filtered for a particular victim net:

```
si_filter_per_aggr_xcap
si_filter_per_aggr_xcap_to_gcap_ratio
si_filter_per_aggr_to_average_aggr_xcap_ratio
```

NanoTime evaluates an aggressor net for possible filtering based on the conditions described in the "Filtering Criteria" section. A filtered aggressor net can still be considered as an aggressor net to other victim nets. Note that a coupling capacitor can be filtered at one end and not at the other. You disable this type of filtering by setting the **si_filter_per_aggr_xcap** variable to zero.

If the **si_aggressive_filtering_per_aggr** is set to **true**, satisfying any one of the criteria allows this net to be filtered. If the variable is **false** (the default), all conditions must be met.

Filtering Criteria

The following text defines the filtering criteria.

Define the ground capacitance of each subnode of a victim V as

```
Cg(V)[i]
where i = from 1 to Ng(V)
```

Then, the total ground capacitance of V is

```
Cg(V) = Cg(V)[1] + Cg(V)[2] + ... + Cg(V)[Ng(V)]
```

Further, define the coupling capacitors between a subnode of victim V and a subnode of its aggressor j Aj as

```
Cc(V, Aj)[k]
where j = from 1 to Na(V)and k = from 1 to Nc(V, Aj)
```

Then, the total coupling capacitance between V and Aj can be written as

```
Cc(V, Aj) = Cc(V, Aj)[1] + Cc(V, Aj)[2] + ... + Cc(V, Aj)[Nc(V, Aj)]
```

and the total coupling capacitance of V to all its aggressors can be written as

```
Cc(V) = Cc(V, A1) + Cc(V, A2) + ... + Cc(V, AK)
where K is Na(V)
```

The filtering conditions are as follows:

- `Cc(V, Aj) is less than the value of the` **`si_filter_per_aggr_xcap`** `variable.`
- `Cc(V, Aj)/(Cg(V)+Cc(V)) is less than the value of the` **`si_filter_per_aggr_xcap_to_gcap_ratio`** `variable.`
- `Cc(V, Aj)*Na(V)/Cc(V) is less than the value of the` **`si_filter_per_aggr_to_average_aggr_xcap_ratio`** `variable.`

SEE ALSO

```
si_enable_analysis(3)
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)
si_filter_per_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_per_aggr(3)
si_filter_total_aggr_xcap(3)
si_filter_total_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_total(3)
```

si_filter_per_aggr_xcap_to_gcap_ratio

Specifies the minimum value of the ratio of the total cross-coupled capacitance between an aggressor net and a victim net to the total ground capacitance of the victim net, below which the aggressor net can be filtered during parasitic filtering.

TYPE

float

DEFAULT

0.0

DESCRIPTION

Specifies the threshold, in library units, of the ratio of the total cross-coupled capacitance between an aggressor net and a victim net to the total ground capacitance of the victim net.

NanoTime signal integrity analysis uses three variables (including this one) during the second stage of parasitic filtering to determine whether an aggressor net can be filtered for a particular victim net:

```
si_filter_per_aggr_xcap
si_filter_per_aggr_xcap_to_gcap_ratio
si_filter_per_aggr_to_average_aggr_xcap_ratio
```

NanoTime evaluates an aggressor net for possible filtering based on the conditions described in the section "Filtering Criteria." A filtered aggressor net can still be considered as an aggressor net to other victim nets. Note that a coupling capacitor can be filtered at one end and not at the other. You disable this type of filtering by setting the **si_filter_per_aggr_xcap** variable to zero.

If the **si_aggressive_filtering_per_aggr** is set to **true**, satisfying any one of the three conditions allows this net to be filtered. If the variable is **false** (the default), all conditions must be met.

Filtering Criteria

Define the ground capacitance of each subnode of a victim V as

```
Cg(V)[i]
where i = from 1 to Ng(V)
```

Then, the total ground capacitance of V is

```
Cg(V) = Cg(V)[1] + Cg(V)[2] + ... + Cg(V)[Ng(V)]
```

Further, define the coupling capacitors between a subnode of victim V and a subnode of its aggressor j Aj as

```
Cc(V, Aj)[k]
where j = from 1 to Na(V) and k = from 1 to Nc(V, Aj)
```

Then, the total coupling capacitance between V and Aj is

```
Cc(V, Aj) = Cc(V, Aj)[1] + Cc(V, Aj)[2] + ... + Cc(V, Aj)[Nc(V, Aj)]
```

and the total coupling capacitance of V to all its aggressors is

```
Cc(V) = Cc(V, A1) + Cc(V, A2) + ... + Cc(V, AK)
where K is Na(V)
```

The filtering conditions are as follows:

- `Cc(V, Aj) is less than the value of the` **`si_filter_per_aggr_xcap`** `variable.`
- `Cc(V, Aj)/(Cg(V)+Cc(V)) is less than the value of the` **`si_filter_per_aggr_xcap_to_gcap_ratio`** `variable.`
- `Cc(V, Aj)*Na(V)/Cc(V) is less than the value of the` **`si_filter_per_aggr_to_average_aggr_xcap_ratio`** `variable.`

SEE ALSO

```
si_enable_analysis(3)
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)
si_filter_per_aggr_xcap(3)
si_aggressive_filtering_per_aggr(3)
si_filter_total_aggr_xcap(3)
si_filter_total_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_total(3)
```

si_filter_total_aggr_xcap

Specifies the minimum value of the total cross-coupled capacitance between a victim net and all of its aggressor nets, below which the victim net can be filtered during parasitic filtering.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_filter_total_aggr_xcap** variable specifies the threshold, in technology units, of the total cross-coupled capacitance between a victim net and all of its aggressor nets.

NanoTime signal integrity analysis uses two variables (including this one) during the first stage of parasitic filtering to determine whether a victim net can be filtered:

```
si_filter_total_aggr_xcap
si_filter_total_aggr_xcap_to_gcap_ratio
```

NanoTime evaluates a victim net for possible filtering based on the conditions described in the "Filtering Criteria" section. A filtered victim net can still be considered as an aggressor net. Note that a coupling capacitor can be filtered at one end and not at the other. Disable this type of filtering by setting the **si_filter_total_aggr_xcap** variable to zero.

If the **si_aggressive_filtering_total** variable is set to **true**, satisfying any one of the conditions allows the net to be filtered. If the variable is **false** (the default), all conditions must be met.

If a victim net does not meet the criteria, the victim net is not filtered and NanoTime proceeds to the second stage of parasitic filtering. In the second stage, NanoTime evaluates the aggressor nets for this victim net based on the **si_filter_per_aggr_xcap**, **si_filter_per_aggr_xcap_to_gcap_ratio**, and

si_filter_per_aggr_to_average_aggr_xcap_ratio variables.

Filtering Criteria

Define the ground capacitance of each subnode of a victim V as

```
Cg(V)[i]
where i = from 1 to Ng(V)
```

Then, the total ground capacitance of V is

```
Cg(V) = Cg(V)[1] + Cg(V)[2] + ... + Cg(V)[Ng(V)]
```

Further, define the coupling capacitors between a subnode of victim V and a subnode of its aggressor j Aj as

```
Cc(V, Aj)[k]
where j = from 1 to Na(V) and k = from 1 to Nc(V, Aj)
```

Then, the total coupling capacitance between V and Aj is

```
Cc(V, Aj) = Cc(V, Aj)[1] + Cc(V, Aj)[2] + ... + Cc(V, Aj)[Nc(V, Aj)]
```

The total coupling capacitance of V to all its aggressors is

```
Cc(V) = Cc(V, A1) + Cc(V, A2) + ... + Cc(V, AK)
where K = Na(V)
```

The filtering conditions are as follows:

- `Cc(V) is less than the value of the` **si_filter_total_aggr_xcap** `variable.`
- `Cc(V)/(Cg(V)+Cc(V)) is less than the value of the` **si_filter_total_aggr_xcap_to_gcap_ratio** `variable.`

SEE ALSO

```
si_enable_analysis(3)
si_filter_total_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_total(3)
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)
si_filter_per_aggr_xcap(3)
si_filter_per_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_per_aggr(3)
```

si_filter_total_aggr_xcap_to_gcap_ratio

Specifies the minimum value of the ratio of the total cross-coupled capacitance between a victim net and all of its aggressor nets to the total ground and cross-coupled capacitance of the victim net, below which a victim net can be filtered during parasitic filtering.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_filter_total_aggr_xcap_to_gcap_ratio** variable specifies the threshold of the ratio of total cross-coupled capacitance to the total ground and cross-coupled capacitance of a victim net.

NanoTime signal integrity analysis uses two variables (including this one) during the first stage of parasitic filtering to determine whether a victim net can be filtered:

```
si_filter_total_aggr_xcap
si_filter_total_aggr_xcap_to_gcap_ratio
```

NanoTime evaluates a victim net for possible filtering based on the conditions described in the section "Filtering Criteria." A filtered victim net can still be considered as an aggressor net. Note that a coupling capacitor can be filtered at one end and not at the other. You disable this type of filtering by setting the **si_filter_total_aggr_xcap** variable to zero.

If the **si_aggressive_filtering_total** variable is set to **true** then satisfying any one of the conditions allows the net to be filtered. If the variable is **false** (the default), then all conditions must be met.

If a victim net does not meet the criteria, the victim net is not filtered and NanoTime proceeds to the second stage of parasitic filtering. In the second stage, NanoTime evaluates the aggressor nets for this

victim net based on the **si_filter_per_aggr_xcap**, **si_filter_per_aggr_xcap_to_gcap_ratio**, and **si_filter_per_aggr_to_average_aggr_xcap_ratio** variables.

Filtering Criteria

Define the ground capacitance of each subnode of a victim V as

```
Cg(V)[i]
where i = from 1 to Ng(V)
```

Then the total ground capacitance of V is

```
Cg(V) = Cg(V)[1] + Cg(V)[2] + ... + Cg(V)[Ng(V)]
```

Further, define the coupling capacitors between a subnode of victim V and a subnode of its aggressor j Aj as

```
Cc(V, Aj)[k]
where j = from 1 to Na(V), and k = from 1 to Nc(V, Aj)
```

Then the total coupling capacitance between A and Vj is

```
Cc(V, Aj) = Cc(V, Aj)[1] + Cc(V, Aj)[2] + ... + Cc(V, Aj)[Nc(V, Aj)]
```

and the total coupling capacitance of V to all its aggressors is

```
Cc(V) = Cc(V, A1) + Cc(V, A2) + ... + Cc(V, AK)
where K = Na(V)
```

The filtering conditions are as follows:

- `Cc(V) is less than the value of the` **si_filter_total_aggr_xcap** `variable.`
- `Cc(V)/(Cg(V)+Cc(V)) is less than the value of the` **si_filter_total_aggr_xcap_to_gcap_ratio** `variable.`

SEE ALSO

```
si_enable_analysis(3)
si_filter_total_aggr_xcap(3)
si_aggressive_filtering_total(3)
si_filter_per_aggr_to_average_aggr_xcap_ratio(3)
si_filter_per_aggr_xcap(3)
si_filter_per_aggr_xcap_to_gcap_ratio(3)
si_aggressive_filtering_per_aggr(3)
```

si_move_miller_caps_into_fets

Allows reduced pessimism for handling certain coupling capacitors by moving them into timing simulations as capacitors rather than accounting for them as SI aggressors.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime normally accounts for the timing effect of coupling capacitors, capacitors between signal lines, by treating them as SI aggressors. This results in some pessimism, since SI analysis is a pessimistic bounding analysis.

For some coupling capacitors, those which connect pairs of logical nets which are connected to the pins of the same transistor, for example the gate and drain of one transistor, NanoTime can reduce this pessimism.

If you set si_move_miller_caps_into_fets to true, NanoTime does this by moving the effect of this capacitance into the simulation of the transistor, and incorporates its effect in every simulation which includes the transistor. This uses the real waveforms on the transistor pins, and removes the pessimistic approximation inherent in the SI analysis for these capacitances.

SEE ALSO

si_enable_analysis(3)

si_noise_margin_above_high

Specifies the amount of voltage allowed for noise values above the supply rail during signal integrity noise analysis.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_noise_margin_above_high** variable sets the amount of noise voltage allowed above the supply rail before being considered a violation. It is specified in volts.

By default, the margin is set to zero indicating that no noise is allowed. Specify a positive value for a less restrictive check.

The **si_noise_margin_above_high** variable applies to all design objects during signal integrity noise analysis. To override the global margins and define noise margins on specific objects (pins, ports or nets), use the **set_noise_margin** command.

SEE ALSO

```
update_noise(2)
report_noise(2)
si_noise_margin_above_low(3)
```

```
si_noise_margin_below_high(3)
si_noise_margin_below_low(3)
set_noise_margin(2)
```

si_noise_margin_above_low

Specifies the amount of voltage allowed for noise values above the ground rail during signal integrity noise analysis.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_noise_margin_above_low** variable sets the amount of noise voltage above the ground rail allowed before being considered a violation. It is specified in volts.

By default, the margin is set to zero indicating that no noise is allowed. Specify a positive value for a less restrictive check.

The **si_noise_margin_above_low** variable applies to all design objects during signal integrity noise analysis. To override the global margins and define noise margins on specific objects (pins, ports or nets), use the **set_noise_margin** command.

SEE ALSO

```
update_noise(2)
report_noise(2)
si_noise_margin_above_high(3)
```

```
si_noise_margin_below_high(3)
si_noise_margin_below_low(3)
set_noise_margin(2)
```

si_noise_margin_below_high

Specifies the amount of voltage allowed for noise values below the supply rail during signal integrity noise analysis.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_noise_margin_below_high** variable sets the amount of noise voltage below the supply rail allowed before being considered a violation. It is specified in volts.

By default, the margin is set to zero indicating that no noise is allowed. Specify a positive value for a less restrictive check.

The **si_noise_margin_below_high** variable applies to all design objects during signal integrity noise analysis. To override the global margins and define noise margins on specific objects (pins, ports or nets), use the **set_noise_margin** command.

SEE ALSO

```
update_noise(2)
report_noise(2)
si_noise_margin_above_high(3)
```

```
si_noise_margin_above_low(3)
si_noise_margin_below_low(3)
set_noise_margin(2)
```

si_noise_margin_below_low

Specifies the amount of voltage allowed for noise values below the ground rail during signal integrity noise analysis.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_noise_margin_below_low** variable sets the amount of noise voltage below the ground rail allowed before being considered a violation. It is specified in volts.

By default, the margin is set to zero indicating that no noise is allowed. Specify a positive value for a less restrictive check.

The **si_noise_margin_below_low** variable applies to all design objects during signal integrity noise analysis. To override the global margins and define noise margins on specific objects (pins, ports or nets), use the **set_noise_margin** command.

SEE ALSO

```
update_noise(2)
report_noise(2)
si_noise_margin_above_high(3)
```

```
si_noise_margin_above_low(3)
si_noise_margin_below_high(3)
set_noise_margin(2)
```

si_timing_window_overlap_tolerance

Specifies a tolerance that qualifies any two timing windows to be considered overlapping during signal integrity analysis.

TYPE

float

DEFAULT

0.0

DESCRIPTION

The **si_timing_window_overlap_tolerance** variable specifies the amount of margin in user time units where two timing windows are considered to be overlapping. If the timing windows are separated by an amount less than this value, they are considered to be overlapping during signal integrity delay and noise analysis.

In effect, half of the tolerance value is added to the beginning of each timing window and half to the end of the timing window. If the added margin results in a timing window exceeding the clock period, the timing window is set to an infinite window.

SEE ALSO

```
si_enable_analysis(3)
si_enable_noise_analysis(3)
timing_multi_input_overlap_tolerance(3)
```

si_xtalk_delay_analysis_mode

Specifies the arrival window alignment mode for crosstalk delay analysis.

TYPE

string

DEFAULT

all_paths

DESCRIPTION

This variable specifies how timing alignment is performed between victim and aggressors for crosstalk delay analysis. The allowed settings are **all_paths** (the default) and **worst_path**. The default, **all_paths**, causes NanoTime to calculate crosstalk for all paths through the victim net. A value of **worst_path** causes NanoTime to calculate crosstalk only for the worst paths through the victim net. The worst paths are the earliest path and the latest path, which have the smallest or most negative slack.

In **all_paths** alignment mode, NanoTime considers the largest possible crosstalk delta delay for the given victim and aggressor arrival windows. This guarantees a conservative analysis of all paths going through the victim net with different arrival times. However, applying the worst crosstalk delta delay to all paths, including the worst path, is pessimistic because the worst crosstalk delta delay might not result from the same conditions as the worst path. You can avoid this pessimism by performing dynamic simulation of the path in question, but it is not practical to do this for all paths in the design.

In **worst_path** alignment mode, NanoTime aligns aggressors for the earliest/latest paths on the victim, so that only the crosstalk affecting the worst path is considered. As a result, NanoTime calculates only the crosstalk effects that make the latest path slower, or the earliest path faster. The **worst_path** mode typically generates delta delays for fewer paths, reducing crosstalk effects, allowing for faster convergence and reduced pessimism. This approach ensures an accurate analysis of the worst paths.

However, the **worst_path** alignment mode considers crosstalk effects only for the worst paths, possibly allowing inaccurate analysis of crosstalk delay effects for subcritical (non-worst) paths. As a result, if you

report more than one path by using the **-nworst N** option of the **report_paths** command with N>1, NanoTime might report optimistic slacks for subcritical paths. Also, the **report_constraint** command with the **-all_violators -max_delay -min_delay** options might report fewer violations than really exist in the design if subcritical paths have very small positive slack values and crosstalk effects are missed.

SEE ALSO

```
report_paths(2)
si_enable_analysis(3)
```

si_xtalk_exit_on_coupled_reevaluated_nets_pc

Specifies the percentage of nets selected for reevaluation, relative to the total number of coupled nets, below which NanoTime exits the signal integrity analysis loop.

TYPE

float

DEFAULT

0

DESCRIPTION

The **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable directs NanoTime to exit the signal integrity analysis loop based on the percentage of nets selected for reevaluation in the next iteration relative to the total number of coupled nets.

If the percentage of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable, NanoTime exits the analysis loop after completing the current iteration. The number of coupled nets is based on detailed parasitics as read in by the **read_parasitics** command. In other words, crosstalk filtering does not affect the count of coupled nets for this purpose. The number of coupled nets includes all individual net segments in the same way that a [get_nets -hierarchical *] command counts all nets in the design.

This variable is one of six variables that determine the exit criteria of signal integrity analysis iterations. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **si_xtalk_exit_on_max_iteration_count** variable.
2. All delta delays fall between the values of the **si_xtalk_exit_on_min_delta_delay** and **si_xtalk_exit_on_max_delta_delay** variables.

3. The number of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_number_of_reevaluated_nets** variable.
4. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_reevaluated_nets_pct** variable.
5. The percentage of nets (relative to the number of cross-coupled nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable.
6. You manually exit the analysis loop by pressing Control+C to send an interrupt signal to the NanoTime process. The interrupt is handled as any other exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting NanoTime.

SEE ALSO

```
si_xtalk_exit_on_max_delta_delay(3)
si_xtalk_exit_on_max_iteration_count(3)
si_xtalk_exit_on_min_delta_delay(3)
si_xtalk_exit_on_number_of_reevaluated_nets(3)
si_xtalk_exit_on_reevaluated_nets_pct(3)
```

si_xtalk_exit_on_max_delta_delay

Specifies the upper bound of a window of delta delay values, within which NanoTime exits the signal integrity analysis loop.

TYPE

float

DEFAULT

0

DESCRIPTION

The **si_xtalk_exit_on_max_delta_delay** variable directs NanoTime to exit the signal integrity analysis loop based on the maximum delta delay value, in library delay units.

In signal integrity analysis, the delta delay for a net is the change in the delay from one iteration to the next. If the delta delay values for all nets fall within the window defined by the **si_xtalk_exit_on_max_delta_delay** and **si_xtalk_exit_on_min_delta_delay** variables, NanoTime exits the signal integrity analysis loop after completing the current iteration.

As an example, consider the values of these two variables as follows:

```
set si_xtalk_exit_on_min_delta_delay -1.0
set si_xtalk_exit_on_max_delta_delay 2.0
```

For a design that has delta delay values of {-3, 1.5, 1.7}, the value -3 falls outside the window (-1.0 to 2.0), so NanoTime does not exit the analysis loop. However, if in the next iteration the delay of -3 improves to -0.9, all delta delays fall within the window and NanoTime exits the analysis loop.

This variable is one of six variables that determine the exit criteria of signal integrity analysis iterations. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **si_xtalk_exit_on_max_iteration_count** variable.
2. All delta delays fall between the values of the **si_xtalk_exit_on_min_delta_delay** and **si_xtalk_exit_on_max_delta_delay** variables.
3. The number of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_number_of_reevaluated_nets** variable.
4. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_reevaluated_nets_pct** variable.
5. The percentage of nets (relative to the number of cross-coupled nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable.
6. You manually exit the analysis loop by pressing Control+C to send an interrupt signal to the NanoTime process. The interrupt is handled as any other exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting NanoTime.

SEE ALSO

si_xtalk_exit_on_max_iteration_count(3)
si_xtalk_exit_on_min_delta_delay(3)
si_xtalk_exit_on_number_of_reevaluated_nets(3)
si_xtalk_exit_on_reevaluated_nets_pct(3)
si_xtalk_exit_on_coupled_reevaluated_nets_pct(3)

si_xtalk_exit_on_max_iteration_count

Specifies the maximum number of signal integrity analysis iterations, after which NanoTime exits the analysis loop.

TYPE

integer

DEFAULT

3

DESCRIPTION

The **si_xtalk_exit_on_max_iteration_count** variable specifies the maximum number of incremental timing iterations during signal integrity delay analysis. NanoTime exits the analysis loop after performing this number of iterations. The default is 3 and the minimum is 1.

This variable is one of six variables that determine the exit criteria of signal integrity analysis iterations. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **si_xtalk_exit_on_max_iteration_count** variable.
2. All delta delays fall between the values of the **si_xtalk_exit_on_min_delta_delay** and **si_xtalk_exit_on_max_delta_delay** variables.
3. The number of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_number_of_reevaluated_nets** variable.
4. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_reevaluated_nets_pct** variable.

5. The percentage of nets (relative to the number of cross-coupled nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable.

6. You manually exit the analysis loop by pressing Control+C to send an interrupt signal to the NanoTime process. The interrupt is handled as any other exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting NanoTime.

SEE ALSO

```
si_xtalk_exit_on_max_delta_delay(3)
si_xtalk_exit_on_min_delta_delay(3)
si_xtalk_exit_on_number_of_reevaluated_nets(3)
si_xtalk_exit_on_reevaluated_nets_pct(3)
si_xtalk_exit_on_coupled_reevaluated_nets_pct(3)
```

si_xtalk_exit_on_min_delta_delay

Specifies the lower bound of a window of delta delay values, within which NanoTime exits the signal integrity analysis loop.

TYPE

float

DEFAULT

0

DESCRIPTION

The **si_xtalk_exit_on_min_delta_delay** variable directs NanoTime to exit the signal integrity analysis loop based on the minimum delta delay value, in library delay units.

In signal integrity analysis, the delta delay for a net is the change in the delay from one iteration to the next. If the delta delay values for all nets fall within the window defined by the **si_xtalk_exit_on_max_delta_delay** and **si_xtalk_exit_on_min_delta_delay** variables, NanoTime exits the signal integrity analysis loop after completing the current iteration.

As an example, consider the values of these two variables as follows:

```
set si_xtalk_exit_on_min_delta_delay -1.0
set si_xtalk_exit_on_max_delta_delay 2.0
```

For a design that has delta delay values of {-3, 1.5, 1.7}, the value -3 falls outside the window (-1.0 to 2.0), so NanoTime does not exit the analysis loop. However, if in the next iteration the delay of -3 improves to -0.9, all delta delays fall within the window and NanoTime exits the analysis loop.

This variable is one of six variables that determine the exit criteria of signal integrity analysis iterations. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **si_xtalk_exit_on_max_iteration_count** variable.
2. All delta delays fall between the values of the **si_xtalk_exit_on_min_delta_delay** and **si_xtalk_exit_on_max_delta_delay** variables.
3. The number of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_number_of_reevaluated_nets** variable.
4. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_reevaluated_nets_pct** variable.
5. The percentage of nets (relative to the number of cross-coupled nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable.
6. You manually exit the analysis loop by pressing Control+C to send an interrupt signal to the NanoTime process. The interrupt is handled as any other exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting NanoTime.

SEE ALSO

si_xtalk_exit_on_max_delta_delay(3)
si_xtalk_exit_on_max_iteration_count(3)
si_xtalk_exit_on_number_of_reevaluated_nets(3)
si_xtalk_exit_on_reevaluated_nets_pct(3)
si_xtalk_exit_on_coupled_reevaluated_nets_pct(3)

si_xtalk_exit_on_number_of_reevaluated_nets

Specifies the maximum number of nets selected for reevaluation, below which NanoTime exits the signal integrity analysis loop.

TYPE

integer

DEFAULT

0

DESCRIPTION

The **si_xtalk_exit_on_number_of_reevaluated_nets** variable directs NanoTime to exit the signal integrity analysis loop based on the maximum number of nets selected for reevaluation in the next iteration.

If the number of nets selected for reevaluation in the next iteration is less than this number, NanoTime exits the analysis loop after completing the current iteration.

This variable is one of six variables that determine the exit criteria of signal integrity analysis iterations. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **si_xtalk_exit_on_max_iteration_count** variable.
2. All delta delays fall between the values of the **si_xtalk_exit_on_min_delta_delay** and **si_xtalk_exit_on_max_delta_delay** variables.
3. The number of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_number_of_reevaluated_nets** variable.

4. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_reevaluated_nets_pct** variable.
5. The percentage of nets (relative to the number of cross-coupled nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable.
6. You manually exit the analysis loop by pressing Control+C to send an interrupt signal to the NanoTime process. The interrupt is handled as any other exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting NanoTime.

SEE ALSO

```
si_xtalk_exit_on_max_delta_delay(3)
si_xtalk_exit_on_max_iteration_count(3)
si_xtalk_exit_on_min_delta_delay(3)
si_xtalk_exit_on_reevaluated_nets_pct(3)
si_xtalk_exit_on_coupled_reevaluated_nets_pct(3)
```

si_xtalk_exit_on_reevaluated_nets_pct

Specifies the maximum percentage of nets selected for reevaluation relative to the total number of nets, below which NanoTime exits the signal integrity analysis loop.

TYPE

float

DEFAULT

0

DESCRIPTION

The **si_xtalk_exit_on_reevaluated_nets_pct** variable directs NanoTime to exit the signal integrity analysis loop based on the maximum percentage of nets reselected for evaluation, relative to the total number of nets.

If the percentage of nets selected for reevaluation in the next iteration is less than this number, NanoTime exits the analysis loop after completing the current iteration.

This variable is one of six variables that determine the exit criteria of signal integrity analysis iterations. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **si_xtalk_exit_on_max_iteration_count** variable.
2. All delta delays fall between the values of the **si_xtalk_exit_on_min_delta_delay** and **si_xtalk_exit_on_max_delta_delay** variables.
3. The number of nets selected for reevaluation in the next iteration is less than the value of the **si_xtalk_exit_on_number_of_reevaluated_nets** variable.

4. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_reevaluated_nets_pct** variable.
5. The percentage of nets (relative to the number of cross-coupled nets) selected for reevaluation is less than the value of the **si_xtalk_exit_on_coupled_reevaluated_nets_pct** variable.
6. You manually exit the analysis loop by pressing Control+C to send an interrupt signal to the NanoTime process. The interrupt is handled as any other exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting NanoTime.

SEE ALSO

```
si_xtalk_exit_on_max_delta_delay(3)
si_xtalk_exit_on_max_iteration_count(3)
si_xtalk_exit_on_min_delta_delay(3)
si_xtalk_exit_on_number_of_reevaluated_nets(3)
si_xtalk_exit_on_coupled_reevaluated_nets_pct(3)
```

si_xtalk_reselect_delta_delay

Specifies the threshold of the delay change for a net during one iteration of signal integrity (crosstalk) delay analysis, above which NanoTime reselects a net for subsequent signal integrity delay calculations.

TYPE

float

DEFAULT

0

DESCRIPTION

NanoTime uses the results of one iteration of signal integrity delay analysis to determine whether to reselect nets for analysis during the next iteration. The two variables that control net reselection are:

```
si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
```

The **si_xtalk_reselect_delta_delay** variable sets a threshold in terms of absolute delta delay, where delta delay is the change in total delay between analysis iterations. Nets that have at least one net arc with a delta delay above this threshold are selected for the next iteration of crosstalk delay analysis.

The **si_xtalk_reselect_delta_delay_ratio** variable sets a threshold in terms of the delta delay ratio. Nets that have at least one net arc where the ratio of the delta delay to the total delay is above this threshold are selected for the next iteration of crosstalk delay analysis.

SEE ALSO

```
si_enable_analysis(3)
si_xtalk_reselect_delta_delay_ratio(3)
```

si_xtalk_reselect_delta_delay_ratio

Specifies the threshold of the ratio of the delay change for a net during one iteration of signal integrity (crosstalk) delay analysis to the total delay for that net, above which NanoTime reselects a net for subsequent signal integrity delay calculations.

TYPE

float

DEFAULT

0

DESCRIPTION

NanoTime uses the results of one iteration of signal integrity delay analysis to determine whether to reselect nets for analysis during the next iteration. The two variables that control net reselection are:

```
si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
```

The **si_xtalk_reselect_delta_delay** variable sets a threshold in terms of absolute delta delay, where delta delay is the change in total delay between analysis iterations. Nets that have at least one net arc with a delta delay above this threshold are selected for the next iteration of crosstalk delay analysis.

The **si_xtalk_reselect_delta_delay_ratio** variable sets a threshold in terms of the delta delay ratio. Nets that have at least one net arc where the ratio of the delta delay to the total delay is above this threshold are selected for the next iteration of crosstalk delay analysis.

If a net has multiple stage delays (for example, because of a net fanout greater than one or multiple cell arcs), NanoTime considers the delta delay and total delay values that result in the highest delta to stage delay ratio, thus making net reselection conservative.

SEE ALSO

si_enable_analysis(3)
si_xtalk_reselect_delta_delay(3)

sim_cfg_delay_calculation_limit

Specifies the maximum amount of time that NanoTime allows for the simulation of a single stage, in nanoseconds.

TYPE

float

DEFAULT

20

DESCRIPTION

To save runtime, NanoTime stops path searching when the simulation time of a single stage exceeds the limit specified by this variable. The default is 20 nanoseconds.

A stage is one simulation unit, typically a channel-connected block. Set the simulation time limit higher for more accurate calculation of longer delays, or lower to reduce runtime. Be sure that the time is longer than the full transition time at the output of any single stage.

SEE ALSO

`sim_cfg_miller_effect(3)`
`sim_cfg_spd(3)`

sim_cfg_miller_effect

Determines whether NanoTime considers the Miller effect during delay calculation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable determines whether NanoTime considers the Miller effect during delay calculation. The default is **true**, which enables consideration of the Miller effect. If you know that the Miller effect is not significant in your circuit, you can save runtime by setting the variable to **false**.

The Miller effect is a change in effective capacitance when the voltages at the two terminals of a capacitor change at the same time. The coupling capacitance between the terminals of a transistor can have a large effect on delay due to the Miller effect, especially when the parasitic capacitance due to overlap and diffusion dominate the total capacitance at a node and the slope of the input signal is very steep.

SEE ALSO

`sim_cfg_delay_calculation_limit(3)`
`sim_cfg_spd(3)`

sim_cfg_spd

Specifies a parameter that trades off speed and accuracy for the delay calculator.

TYPE

float

DEFAULT

0.05

DESCRIPTION

This variable is a parameter that specifies the tradeoff between the accuracy and the speed of the NanoTime delay calculator. Use a floating-point value between 0.05 and 5, where 0.05 results in the highest accuracy and 5.0 results in the highest speed of execution.

The best value to use depends on the technology. Smaller geometries and smaller parasitic RC elements require smaller variable settings to get high accuracy. The default of 0.05 gives the best accuracy but requires the most runtime.

SEE ALSO

```
sim_cfg_miller_effect(3)
sim_cfg_delay_calculation_limit(3)
```

sim_file_cleanup

Specifies how to clean up the simulation files.

TYPE

integer

DEFAULT

4

DESCRIPTION

This variable specifies how to clean up intermediate simulation files. There are 7 cleanup options; the valid values of the variable are 0 to 6. The following list shows the values and their corresponding cleanup operations. The default is 4.

- 0 Keep all simulation files.
- 1 Keep all files in gzip-compressed format.
- 2 Keep only the spice decks.
- 3 Do not keep any simulation files.
- 4 Keep only files of failed simulations.
- 5 Keep the spice decks in gzip-compressed format.
- 6 Keep files of failed simulations in gzip-compressed format.

sim_finesim_keep_run_files

Specifies the retention of intermediate simulation decks used for DCS and DDS when FineSim is enabled.

TYPE

boolean

DEFAULT

false

DESCRIPTION

This variable causes NanoTime to keep run files used for the simulation of DCS and DDS stages when FineSim is selected. The user needs to exercise caution so as not to cause unacceptable disk space consumption.

Some of the DDS simulations may involve the use of data sweeps. In these scenarios the simulation deck will only capture the stimulus from one of the sweeps.

The run files are provided so that potential tool setup issues or design errors can be investigated. They are not intended for measuring path accuracy.

SEE ALSO

```
mark_simulation(2)
set_simulation_attributes(2)
dcs_enable_analysis(3)
dcs_control_header(3)
dcs_enable_analysis(3)
```

```
dcs_number_of_cycles(3)
dcs_enable_detailed_path_reporting(3)
```

sim_finesim_mode

Specifies the mode for running FineSim as the external simulator.

TYPE

string

DEFAULT

spicead

DESCRIPTION

This variable specifies the mode for running FineSim as the external simulator.

The allowed value for this variable is one of the following strings: **spicead**, **spicehd**, **spicemd**, **spicexd**.

For more information about the mode, refer to the FineSim documentation.

SEE ALSO

`set_simulator(2)`
`sim_finesim_option(3)`

sim_finesim_options

Specifies options for running FineSim as the external simulator.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies options for running FineSim as the external simulator.

EXAMPLES

The following command specifies one option for running FineSim as the external simulator.

```
nt_shell> set sim_finesim_options "finesim_resmin=0.01"
```

The following command specifies multiple options for running FineSim as the external simulator.

```
nt_shell> set sim_finesim_options "finesim_resmin=0.01 finesim_resmax=0.02"
```

For more information about options, refer to the FineSim documentation.

SEE ALSO

set_simulator(2)
sim_finesim_mode(3)

sim_hspice_cmd

Specifies the command used to launch HSPICE simulation of SPICE decks.

TYPE

string

DEFAULT

hspice -i <input_deck> -o <listing_file> 2>&1

DESCRIPTION

The **sim_hspice_cmd** variable specifies the command to launch HSPICE simulations.

This variable specifies the operating system command used to launch simulations required by the following NanoTime commands and features:

- The **set_simulator** command
- The **extract_model -hspice_timing** command
- CCS noise model creation

NanoTime uses the information in the **sim_hspice_cmd** variable to launch HSPICE simulations of the SPICE decks that were created inside NanoTime. This variable does not affect the simulation of SPICE decks created by the **write_spice** command. Use the **write_spice_sim_cmd** variable to specify the command to launch those simulations.

The **<input_deck>** and **<listing_file>** tags are syntax terms and must be written *exactly* as specified, including the angle brackets. Do not change the wording inside the angle brackets. The NanoTime tool replaces the tags with values as follows:

- The **<input_deck>** tag is replaced with one of the SPICE deck file names. For example, a file name

might be abc_00003.sp.

- The **<listing_file>** tag is replaced with the root name of the SPICE deck file name. For example, the root name is abc_00003 for file abc_00003.sp.

If the **sim_hspice_cmd** variable is the default, HSPICE Advanced Client/Server mode will be used for the **extract_model -hspice_timing** command.

EXAMPLES

The following command specifies a user-defined HSPICE wrapper.

```
nt_shell> set sim_hspice_cmd "myhspice <input_deck> -output <listing_file> -w "
```

SEE ALSO

set_simulator(2)
write_spice_sim_cmd(3)

sim_hspice_keep_run_files

Specifies the retention of intermediate HSPICE simulation decks when **set_simulator** is applied.

TYPE

boolean

DEFAULT

false

DESCRIPTION

This variable causes NanoTime to keep run files used for the simulation of stages when **set_simulator** is invoked to select HSPICE as the external simulator. The user needs to exercise caution so as not to cause unacceptable disk space consumption.

The run files are provided so that potential tool setup issues or design errors can be investigated. They are not intended for measuring path accuracy.

SEE ALSO

`set_simulator(2)`

sim_miller_active_load_enable_fanout_limit

Enables or disables the active load fanout limit option for write_spice and NanoTime simulation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the **sim_miller_active_load_enable_fanout_limit** variable is set to true, NanoTime limits the active miller option to nodes with a fanout no more than the value defined by the **sim_miller_active_load_fanout_limit** variable. This avoids significant slowdown caused by large fanout values.

SEE ALSO

`sim_miller_active_load_fanout_limit(3)`

sim_miller_active_load_fanout_limit

set the fanout limit for active miller.

TYPE

integer

DEFAULT

10

DESCRIPTION

When sim_miller_active_load_enable_fanout_limit set to true, NT will limit the active miller option to nodes with number of fanout no more than the value defined by sim_miller_active_load_fanout_limit. The purpose is to avoid significant slowdown introduced by large number of fanouts.

SEE ALSO

sim_miller_active_load_enable_fanout_limit(3)

sim_miller_direction_check

Enables or disables checking the next switching direction when setting states for active miller loads.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable allows a more detailed heuristic to apply when using active load transistors during delay calculation. By default, active loads assume that their output switches in the opposite direction of the loaded net.

When this variable is set to **true**, the direction of switching on the next net is checked to determine if it is the same as or different from the loaded net. Then logic states and initial conditions are set accordingly.

This method can only be applied to nets driving transmission gate (tgate) topologies that are part of muxes in the design. You can include other tgates by marking their control nets using the **mark_net -check_miller_direction** command.

This variable only has an effect when the **sim_miller_use_active_load** variable is set to **true**.

SEE ALSO

`mark_net(2)`

```
sim_miller_use_active_load(3)
```

sim_miller_limit_for_decoders

Limits the use of active load Miller effect analysis on some side loads to a path through a decoder, when it can be shown that tracing through the path implies that side loads on this path are deselected and cannot switch.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, the deselected side loads are represented either as an inverter disconnected from power on one rail or as two disconnected transistors.

This applies only when the deselected side load is connected to a rail through a single transistor, e.g. through a PMOS transistor in a nand.

SEE ALSO

`sim_miller_use_active_load(3)`

sim_miller_rail_short_detection

Enables the detection and correction of possible rail-rail transient and static short circuits introduced by the addition of active miller transistors to a simulation stage.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime looks for the possibility of short circuits among the load transistors and takes remedial steps. The remedial steps include altering the transistor terminal connections. This is intended to reduce the number of simulation failures caused by shorts.

SEE ALSO

```
sim_miller_active_load_enable_fanout_limit(3)
sim_miller_active_load_fanout_limit(3)
sim_miller_direction_check(3)
sim_miller_use_active_load(3)
sim_miller_use_active_load_min(3)
sim_miller_use_extended_load(3)
si_move_miller_caps_into_fets(3)
```

sim_miller_use_active_load

Enables the active load Miller effect analysis for the **write_spice** command and NanoTime simulations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, the side loads are represented as active transistors and the load Miller effect is captured. By default, this variable is set to **false** and active load is disabled.

During timing model generation, look for the **set_load** command applied to the output port because the second to the last stage in the critical path uses the **set_load** command as part of the active Miller loading for that stage. For example,

--->inv(x1)--->inv(x2)--><(><<)>port>-->Cload

Set a value within the Cload charfile range for the **set_load** command. When the active Miller effect is enabled, the delay on x1, second to the last stage, is dependent on the value set on the output port by the **set_load** command.

The Cload charfile data does not account for the delay of x1, second to the last stage, when active Miller analysis is enabled.

SEE ALSO

sim_miller_use_active_load_min(3)

sim_miller_use_active_load_min

Enables the full active load Miller effect analysis on minimum paths for the **write_spice** command and NanoTime simulations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This global variable applies to minimum delay paths only. Set this variable to **true** only when the **sim_miller_use_active_load** variable is set to **true**. When the **sim_miller_use_active_load_min** variable is **true**, all side loads on minimum delay paths are represented as active transistors and the load Miller effect is captured. By default, this variable is set to **false** and the active load is only enabled for inverter loads.

During timing model generation, look for the **set_load** command applied to the output port because the second to the last stage in the critical path uses the **set_load** command as part of the active Miller loading for that stage. For example,

--->inv(x1)--->inv(x2)--><(><<)>port>-->Cload

Set a value within the Cload charfile range for the **set_load** command. When the active Miller effect is enabled, the delay on x1, second to the last stage, is dependent on the value set on the output port by the **set_load** command.

The Cload charfile data does not account for the delay of x1, second to the last stage, when active Miller analysis is enabled.

SEE ALSO

sim_miller_use_active_load(3)

sim_miller_use_extended_load

Expands the active-Miller chain through the source or drain of the active-load transistor based on transistor direction.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set the **gvar_name** variable to **true** to enhance the accuracy of Miller capacitance modeling, at a cost of additional runtime. The **gvar_name** variable has an effect only if Miller side load analysis is enabled by setting the **sim_miller_use_active_load** variable to **true**.

When the **sim_miller_use_active_load** variable is **true**., Miller side load analysis models a side load circuit as an equivalent inverter with a Miller capacitance load. In a series configuration of transistors, only the switching transistor is modeled in this way. Other transistors in the on-chain are modeled as lumped capacitances.

If you set the **gvar_name** variable to **true**, NanoTime expands the active Miller analysis by replacing the lumped capacitances in the on-chain with real transistors. This replacement occurs only for the adjacent transistors on the downstream side of the active-load transistors (where downstream means in the transistor direction.). However, if NanoTime determines that setting the selected transistor to an on state results in a logic conflict with other connections to the gate of that transistor, the active load model is not applied and the lumped capacitance model is retained for that transistor.

SEE ALSO

sim_miller_use_active_load(3)

sim_side_transistor_pin_load_model

Controls how the loading from certain transistors is modeled.

TYPE

string

DEFAULT

capacitor

DESCRIPTION

Certain source or drain connected transistors load a simulated circuit but do not directly contribute to switching the output of the circuit. For instance, a NAND gate with a side input set high has a pfet gated by that input. This pfet is turned off during the simulation, but its drain capacitance loads the output of the NAND gate.

By default, NanoTime models the load due to this drain capacitance as a linear capacitance. This is selected by setting the **sim_side_transistor_pin_load_model** variable to **capacitor**.

Alternatively, NanoTime can more accurately model the load as a full transistor, including all of the nonlinearities in the capacitance. To select this method, set the **sim_side_transistor_pin_load_model** variable to **transistor**. NanoTime intelligently evaluates where in the design to apply the full_transistor load model. However, setting the variable to **transistor** might significantly increase the runtime.

SEE ALSO

`tech_charge_step_scale_factor(3)`

sim_snps_predriver_mix_ratio

Define the mix ratio of exponential waveform when using Synopsys pre-defined driver waveform to approximate the waveform used for simulation input.

TYPE

float

DEFAULT

0.5

DESCRIPTION

When set to 0, the predriver waveform becomes an exponential waveform. When set to 1, the predriver waveform is a linear ramp.

SEE ALSO

sim_use_snps_predriver(3)

sim_transistor_wrapper_subckts

Specifies subcircuit names which are to be used as wrapper subcircuits in SPICE decks.

TYPE

string

DEFAULT

""

DESCRIPTION

This parameter specifies the names of subcircuits which are preferred during the wrapper subcircuit search. The search proceeds from a transistor up the instance hierarchy to find the highest subcircuit that structurally is acting as a wrapper. The search will terminate early if a subcircuit is found whose name is in the **sim_transistor_wrapper_subckts** name list.

The format for this value is a space-separated list of names.

SEE ALSO

`write_spice(2)`

sim_transition_max_default_ratio

Sets the ratio of simulation transition time to the default transition time, above which a warning message is generated.

TYPE

float

DEFAULT

25

DESCRIPTION

An unexpectedly slow transition time during simulation or specification of primary inputs can indicate potential issues with circuit design or NanoTime setup. A warning message is generated when a simulation does not yield a delay by comparing the transition time against the default transition time. A warning message is also generated if transitions specified for primary inputs are too large compared to the defaults.

SEE ALSO

```
set_input_transition(2)
sim_transition_max_fall(3)
sim_transition_max_rise(3)
sim_transition_min_fall(3)
sim_transition_min_rise(3)
```

sim_transition_max_delay_ratio

Sets the ratio of simulation transition time to the delay, above which a warning message is generated.

TYPE

float

DEFAULT

25

DESCRIPTION

An unexpectedly slow transition time during simulation can indicate potential issues with circuit design or NanoTime setup. A warning message is generated when the ratio of the input simulation transition time to the simulation delay exceeds the ratio set by this global variable.

SEE ALSO

`trace_paths(2)`

sim_transition_max_fall

Specifies the default maximum fall time.

TYPE

float

DEFAULT

0.05

DESCRIPTION

This command specifies the default maximum full-swing fall time, in nanoseconds. This value is used for nodes without assigned or calculated fall times.

SEE ALSO

set_input_transition(2)
sim_transition_max_rise(3)
sim_transition_min_fall(3)
sim_transition_min_rise(3)

sim_transition_max_limit

Specifies the maximum allowable transition time.

TYPE

float

DEFAULT

10

DESCRIPTION

This variable specifies the maximum allowable transition time used for simulation, in nanoseconds. This value overrides any larger value calculated by NanoTime or specified by the user.

SEE ALSO

`set_input_transition(2)`
`sim_transition_min_limit(3)`

sim_transition_max_rise

Specifies the default maximum rise time.

TYPE

float

DEFAULT

0.05

DESCRIPTION

This command specifies the default maximum full-swing rise time, in nanoseconds. This value is used for nodes without assigned or calculated rise times.

SEE ALSO

set_input_transition(2)
sim_transition_max_fall(3)
sim_transition_min_fall(3)
sim_transition_min_rise(3)

sim_transition_min_fall

Specifies the default minimum fall time.

TYPE

float

DEFAULT

0.05

DESCRIPTION

This command specifies the default minimum full-swing fall time, in nanoseconds. This value is used for nodes without assigned or calculated fall times.

SEE ALSO

```
set_input_transition(2)
sim_transition_max_fall(3)
sim_transition_max_rise(3)
sim_transition_min_rise(3)
```

sim_transition_min_limit

Specifies the minimum allowable transition time.

TYPE

float

DEFAULT

0.003

DESCRIPTION

This variable specifies the minimum allowable transition time used for simulation, in nanoseconds. This value overrides any smaller value calculated by NanoTime or specified by the user.

SEE ALSO

`set_input_transition(2)`
`sim_transition_max_limit(3)`

sim_transition_min_rise

Specifies the default minimum rise time.

TYPE

float

DEFAULT

0.05

DESCRIPTION

This command specifies the default minimum full-swing rise time, in nanoseconds. This value is used for nodes without assigned or calculated rise times.

SEE ALSO

set_input_transition(2)
sim_transition_max_fall(3)
sim_transition_max_rise(3)
sim_transition_min_fall(3)

sim_use_snps_predriver

Specifies whether to use Synopsys predefined driver waveforms to approximate the waveforms used for simulation inputs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sim_use_snps_predriver** variable specifies how input waveforms of a predriver are determined for nets not using the nonlinear waveform feature. If the nonlinear waveform feature is enabled with the **set_nonlinear_waveform** command, the **sim_use_snps_predriver** variable has no effect for those nets.

The default of the **sim_use_snps_predriver** variable is **false**, which means that two-point ramps are used as simulation inputs. When the variable is set to **true**, the Synopsys predefined driver waveform is used.

The Synopsys predefined driver waveform is a mixture of an exponential waveform and a linear ramp. The default mixing ratio is 0.5. You can change the mixing ratio by using the **sim_snps_predriver_mix_ratio** variable.

SEE ALSO

```
    set_nonlinear_waveform(2)
    sim_snps_predriver_mix_ratio(3)
```

soi_enable_analysis

Enables silicon-on-insulator (SOI) analysis using NanoTime Ultra.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, the **soi_enable_analysis** variable enables analysis using silicon-on-insulator (SOI) transistors. A NanoTime Ultra license is required to use this feature.

SEE ALSO

```
report_technology(2)
set_soi_parameters(2)
set_soi_transistor_type(2)
```

synopsys_program_name

Indicates the name of the program currently running.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the name of the program you are running. This is useful when writing scripts that are mostly common between some applications, but contain some differences based on the application.

To determine the current value of this variable, use **get_app_var synopsys_program_name**.

SEE ALSO

get_app_var(2)

synopsys_root

Indicates the root directory from which the application was run.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the root directory from which the application was run.

To determine the current value of this variable, use **get_app_var synopsys_root**.

SEE ALSO

get_app_var(2)

tcl_interactive

Tcl variable

TYPE

integer

DEFAULT

leave at default setting

DESCRIPTION

This is a Tcl variable, not to be changed by the user.

SEE ALSO

none

tcl_library

Tcl variable

TYPE

string

DEFAULT

leave at default setting

DESCRIPTION

This is a Tcl variable, not to be changed by the user.

SEE ALSO

none

tcl_pkgPath

Tcl variable

TYPE

string

DEFAULT

leave at default setting

DESCRIPTION

This is a Tcl variable, not to be changed by the user.

SEE ALSO

none

tech_charge_step_scale_factor

Scales the voltage steps to compute charge for Level-72 MOSFETs.

TYPE

float

DEFAULT

2.0

DESCRIPTION

Scale the voltage steps to compute charge for Level-72 MOSFETs. The valid range is 1.0 to 3.0, both numbers inclusive. If the value is specified as larger than 1, NanoTime interpolates the values if necessary. By increasing this number, the **check_design** runtime will be decreased.

SEE ALSO

`check_design(2)`

tech_default_voltage

Specifies the power supply voltage to use for transistor models that do not include a voltage specification.

TYPE

float

DEFAULT

5.0

DESCRIPTION

The **tech_default_voltage** variable specifies a default power supply voltage to use when a SPICE model file does not include a supply voltage directive.

To view a list of the current technology registry entries and their associated power supply voltages, use the **list_technology** command.

SEE ALSO

link_design(2)
list_technology(2)
oc_global_voltage(3)

tech_match_length_pct

Specifies the allowable percentage difference between the length of a transistor and the length of a transistor model used to represent that transistor.

TYPE

float

DEFAULT

1

DESCRIPTION

This variable specifies a threshold for mapping transistors to available transistor models that do not have exactly matching transistor lengths. The default is 1, which requires an exact match between the actual transistor length and the transistor model length.

Set the variable to a value greater than 1 to allow some difference. For example, set it to 2 to allow the model length to differ by up to 2 percent from the actual transistor length.

To find the model that most closely matches a given transistor, NanoTime first considers the length, then the width, and finally the other transistor parameters. The allowable parameter tolerances are set with the following variables:

1. tech_match_length_pct
2. tech_match_width_pct
3. tech_match_param_pct

SEE ALSO

read_techfile(2)
set_technology(2)
tech_match_param_pct(3)
tech_match_width_pct(3)

tech_match_param_pct

Specifies the allowable percentage difference between the parameters of a transistor and the parameters of a transistor model used to represent that transistor (exclusive of length and width).

TYPE

float

DEFAULT

1

DESCRIPTION

This variable specifies a threshold for mapping transistors to available transistor models that do not have exactly matching transistor parameters, exclusive of the length and width parameters. The default is 1, which requires an exact match between the actual transistor parameters and the transistor model parameters.

Set the variable to a value greater than 1 to allow some difference. For example, set it to 2 to allow the model parameters to differ by up to 2 percent from the actual transistor parameters.

To find the model that most closely matches a given transistor, NanoTime first considers the length, then the width, and finally the other transistor parameters. The allowable parameter tolerances are set with the following variables:

1. tech_match_length_pct
2. tech_match_width_pct
3. tech_match_param_pct

SEE ALSO

```
read_techfile(2)
set_technology(2)
tech_match_length_pct(3)
tech_match_width_pct(3)
```

tech_match_width_pct

Specifies the allowable percentage difference between the width of a transistor and the width of a transistor model used to represent that transistor.

TYPE

float

DEFAULT

1

DESCRIPTION

This variable specifies a threshold for mapping transistors to available transistor models that do not have exactly matching transistor widths. The default is 1, which requires an exact match between the actual transistor width and the transistor model width.

Set the variable to a value greater than 1 to allow some difference. For example, set it to 2 to allow the model width to differ by up to 2 percent from the actual transistor width.

To find the model that most closely matches a given transistor, NanoTime first considers the length, then the width, and finally the other transistor parameters. The allowable parameter tolerances are set with the following variables:

1. tech_match_length_pct
2. tech_match_width_pct
3. tech_match_param_pct

SEE ALSO

read_techfile(2)
set_technology(2)
tech_match_param_pct(3)
tech_match_length_pct(3)

tech_netlist_spice_model_name

Specifies the name of the technology registry entry created by the **link_design** command to contain the transistor models in the SPICE deck.

TYPE

string

DEFAULT

typ

DESCRIPTION

By default, if there are SPICE models present in the input SPICE deck, the **link_design** command links the transistors in the netlist to the SPICE transistor models. The **link_design** command creates a technology registry entry to hold the SPICE models and assigns the technology name specified by the **tech_netlist_spice_model_name** variable. The default setting causes the technology to be named "typ".

To view a list of the current technology registry entries, use the **list_technology** command.

To specify whether the **link_design** command links the transistors to the SPICE transistor models, use the **link_enable_netlist_spice_model_linking** variable.

SEE ALSO

```
link_design(2)
list_technology(2)
```

link_enable_netlist_spice_model_linking(3)

tech_pocv_match_length_pct

Specifies the allowable percentage difference between the length of a transistor and the length used in the definition of a variation value from the **set_variation_parameters** command.

TYPE

float

DEFAULT

1

DESCRIPTION

This variable specifies a threshold for mapping transistors to available variation definitions. If a variation value for a transistor has been defined using the **set_variation_parameters** command with a length value, the length of a transistor must be within this percentage of the specified length for the variation value to apply. If the length is not within the threshhold, the variation value is not used and a default variation value is used instead.

SEE ALSO

```
set_variation_parameters(2)
report_variation(2)
tech_pocv_match_voltage_pct(3)
tech_pocv_match_wrapper_parameter_pct(3)
timing_pocv_sigma(3)
```

tech_pocv_match_voltage_pct

Specifies the allowable percentage difference between the voltage of a transistor and the voltage used in the definition of a variation value from the **set_variation_parameters** command.

TYPE

float

DEFAULT

1

DESCRIPTION

This variable specifies a threshold for mapping transistors to available variation definitions that the user has entered. If a variation value for a transistor has been defined using the **set_variation_parameters** command with a voltage value specified, then the voltage of a transistor must be within this percentage of the specified value in order for this value to apply. If the voltage is not within this threshhold then the variation value is not used, and a default value is used instead.

SEE ALSO

```
set_variation_parameters(2)
report_variation(2)
tech_pocv_match_length_pct(3)
tech_pocv_match_wrapper_parameter_pct(3)
timing_pocv_sigma(3)
```

tech_pocv_match_wrapper_parameter_pct

Specifies the allowable percentage difference between the macro model wrapper parameters of a transistor and the parameters used in the definition of a variation value from the **set_variation_parameters** command with **-wrapper_parameter** option.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies a threshold for mapping transistors to available variation definitions. If a variation value for a transistor has been defined using the **set_variation_parameters** command with **-wrapper_parameter** option, the wrapper parameter of a transistor must be within this percentage of the specified value for the variation value to apply. If the value is not within the threshhold, the variation value is not used and a default variation value is used instead.

EXAMPLES

The following commands specify 10% tolerance for wrapper parameter par1 and 15% tolerance for parameter par2 which are used in **set_variation_parameters** commands with **-wrapper_parameter** option.

```
nt_shell> set tech_pocv_match_wrapper_parameter_pct "par1=10 par2=15"
nt_shell> set_variation_parameters -type nmos -width 0.12 -wrapper_parameters {par1=2.2 par2=0.95}
```

```
nt_shell> set_variation_parameters -type nmos -width 0.21 -wrapper_parameters {par2=1.2 par3=12.0}
```

SEE ALSO

set_variation_parameters(2)
report_variation(2)
tech_pocv_match_length_pct(3)
tech_pocv_match_voltage_pct(3)
timing_pocv_sigma(3)

timing_all_clocks_propagated

Determines whether new clocks are created as propagated clocks.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When **timing_all_clocks_propagated** is **true** (the default), all clocks subsequently created by the **create_clock** or **create_generated_clock** command are propagated clocks. When the variable is **false**, new clocks are created as ideal clocks.

For propagated clocking, NanoTime calculates the cumulative delays along the clock network, including the effects of wire parasitics. For ideal clocking, NanoTime uses latency values set on objects with the **set_clock_latency** command, or zero latency where no latency value has been set. In either case, you can change the property of a clock by using the **set_propagated_clock** or **remove_propagated_clock** commands.

Setting this variable only affects new clocks created after the setting is changed. Existing clocks are not affected.

SEE ALSO

`create_clock(2)`

```
create_generated_clock(2)
remove_propagated_clock(2)
set_clock_latency(2)
set_propagated_clock(2)
```

timing_analysis_coverage

Enables the capability to run the **report_analysis_coverage** command after path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime collects data during path tracing for use with the **report_analysis_coverage** command. By default, this variable is set to **false** and analysis coverage data is not collected.

You must set this variable before the **trace_paths** or **extract_model** command.

SEE ALSO

report_analysis_coverage(2)
trace_paths(2)
extract_model(2)

timing_analysis_coverage_fanin

Enables the capability to run enhanced **report_analysis_coverage** and **report_fanin** commands after path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime collects data during path tracing for use with the **report_analysis_coverage** and **report_fanin** commands. It allows for listing detailed untested reasons in **report_analysis_coverage** report, and listing of pins/instances in transitive fanin cone in **report_fanin** report. By default, this variable is set to **false** and detailed analysis coverage data is not collected.

You must set this variable before the **trace_paths** or **extract_model** command. Note that setting this variable true also forces the variable **timing_analysis_coverage** true.

SEE ALSO

```
report_analysis_coverage(2)
report_fanin(2)
trace_paths(2)
extract_model(2)
```

```
timing_analysis_coverage(3)
```

timing_arrival_shift_reference

Specifies whether to exclude the launch clock arrival time from the delay value reported by the **report_arrivals** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable specifies the reference point for the reporting of arrival times by the **report_arrivals** command.

By default, the variable is set to **false**. As a result, reported arrival times include the delay of the clock edge that initiates the launch of data. For example, if the active edge of the clock occurs at time = 2.5 and the path delay from the clock edge to the pin is 3.0, then the arrival at the pin is reported to occur at time = 5.5.

However, if this variable is set to **true**, the reported arrival time includes only the path delay from the clock to the pin, or 3.0 time units rather than 5.5.

To have an effect, the variable must be set before you use the **trace_paths** command.

The variable also affects the attributes stored in the path database and reported by the **get_attribute** command, such as the following:

```
nt_shell> get_attribute [get_pin X13.MP1.g] max_falling_arrival
```

SEE ALSO

```
report_arrivals(2)
trace_paths(2)
```

timing_clock_gate_hold_fall_margin

Specifies the amount of timing margin for falling-edge clock-gating hold checks.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge clock-gating hold checks. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any clock-gating hold check.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_gated_clock_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_gated_clock_timing_check(2)
report_constraint(2)
set_gated_clock_timing_check_attributes(2)
timing_clock_gate_hold_rise_margin(3)
timing_clock_gate_setup_fall_margin(3)
timing_clock_gate_setup_rise_margin(3)
```

timing_clock_gate_hold_rise_margin

Specifies the amount of timing margin for rising-edge clock-gating hold checks.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge clock-gating hold checks. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any clock-gating hold check.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_gated_clock_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_gated_clock_timing_check(2)
report_constraint(2)
set_gated_clock_timing_check_attributes(2)
timing_clock_gate_hold_fall_margin(3)
timing_clock_gate_setup_fall_margin(3)
timing_clock_gate_setup_rise_margin(3)
```

timing_clock_gate_setup_fall_margin

Specifies the amount of timing margin for falling-edge clock-gating setup checks.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge clock-gating setup checks. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any clock-gating setup check.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_gated_clock_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_gated_clock_timing_check(2)
report_constraint(2)
set_gated_clock_timing_check_attributes(2)
timing_clock_gate_hold_rise_margin(3)
timing_clock_gate_hold_fall_margin(3)
timing_clock_gate_setup_rise_margin(3)
```

timing_clock_gate_setup_rise_margin

Specifies the amount of timing margin for rising-edge clock-gating setup checks.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge clock-gating setup checks. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any clock-gating setup check.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_gated_clock_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_gated_clock_timing_check(2)
report_constraint(2)
set_gated_clock_timing_check_attributes(2)
timing_clock_gate_hold_rise_margin(3)
timing_clock_gate_hold_fall_margin(3)
timing_clock_gate_setup_fall_margin(3)
```

timing_default_phasing

Sets same-cycle or next-cycle checking for all timing checks in the design.

TYPE

string

DEFAULT

same_cycle

DESCRIPTION

The **timing_default_phasing** variable specifies whether to use same-cycle or next-cycle checking for all timing checks. If the variable is set to **same_cycle** (the default), NanoTime checks for the arrival of data in the same clock cycle in which the data was launched. If the variable is set to **next_cycle**, NanoTime checks for the arrival of data in the next clock cycle after the data was launched.

You can override this global variable setting for specific objects by using the **set_phasing** command, or for specific paths by using the **set_same_phase_path** or **set_no_same_phase_path** commands. The clock cycle used for data arrival checking is also affected by the **set_multicycle_path** command.

Same-Cycle and Next-Cycle Checking

If the output of a latch has a combinational path to the input of another latch, and if the opening edge of the clock at the upstream (launch) latch occurs at the same time as the opening edge at the downstream (capture) latch, NanoTime assumes by default that the two latches operate in the same phase and that the capture latch is transparent. The default checking behavior under these conditions is called same-cycle checking.

The launch and capture opening edges are considered to occur at the same time when the edge times of the reference clocks (as defined by the **create_clock** command) occur at the same time, without considering differences due to uncertainty, latency, or clock network delay. If the launch and capture

latches have opening edges at different times, but with overlapping "on" times, NanoTime assumes by default that data capture occurs in the next clock cycle rather than the same clock cycle. (This behavior can be changed by setting the **timing_intersection_transparency** variable.)

Under same-cycle checking conditions, for a setup check, data launched at the upstream latch by the opening edge of the launch clock must arrive at the downstream latch before the closing edge of the capture clock in the *same* clock cycle. For a hold check, the same data launched at the upstream latch must arrive at the downstream latch no earlier than the closing edge of the capture clock in the *previous* clock cycle.

Under these conditions, if a downstream latch is designed to capture data in the next clock cycle rather than the same clock cycle, you need to set next-cycle checking so that the timing checks are performed at the correct times. In next-cycle checking, for a setup check, data launched at the upstream latch by the opening edge of the launch clock must arrive at the downstream latch before the closing edge of the capture clock in the *next* clock cycle. For a hold check, the same data launched at the upstream latch must arrive at the downstream latch no earlier than the closing edge of the capture clock in the *same* clock cycle.

There are three ways to specify same-cycle or next-cycle checking:

- the **timing_default_phasing** variable
- the **set_phasing** command
- the **set_same_phase_path** and **set_no_same_phase_path** commands

The **timing_default_phasing** variable sets the default behavior of all timing checks. The **set_phasing** command can be used to override the default behavior for specific objects in the design where sequential timing checks are performed, such as latches, flip-flops, precharge gates, and output ports. The **set_same_phase_path** and **set_no_same_phase_path** commands can be used to override the default behavior for specific timing paths. Path-specific settings have priority over object-based settings.

The timing_default_phasing Variable

To use next-cycle checking throughout the design or for most of the design, set the **timing_default_phasing** variable to **next_cycle**. NanoTime then uses next-cycle checking throughout the design. To override this behavior for specific objects or paths, use the **set_phasing -same_cycle** or the **set_same_phase_path** command.

To use same-cycle checking throughout the design or for most of the design, leave the **timing_default_phasing** variable set to its default setting, **same_cycle**. NanoTime then uses same-cycle checking throughout the design. To override this behavior for specific objects or paths, use the **set_phasing -next_cycle** or **set_no_same_phase_path** command.

The **timing_default_phasing** variable can be changed at any time before you run the **trace_paths** command. To change this variable after running **trace_paths**, reset the path database using the **reset_design -paths** command, then set the variable and run **trace_paths** again.

The set_phasing and remove_phasing Commands

The **set_phasing** command sets the type of checking done at specified instances of objects in the design such as latches, flip-flops, precharge gates, and output ports. These object-based settings override the default behavior set with the **timing_default_phasing** variable.

To set same-cycle checking on objects, use the **set_phasing -same_cycle** command. To set next-cycle checking on objects, use the **set_phasing -next_cycle** command. Use the **-setup** or **-hold**

options to make the setting apply to setup checks only or hold checks only at the object. Otherwise, the setting applies to both setup and hold checks.

You can view the setting on an object by using the **report_attribute** or **get_attribute** commands. Some commands such as **report_port** show the object-based phasing setting.

To remove a setting made on an object, use the **remove_phasing** command.

The set_same_phase_path and set_no_same_phase_path Commands

The **set_same_phase_path** and **set_no_same_phase_path** commands affect the checking of specified timing paths in the design. The **-from**, **-through**, and **-to** options in the command specify the paths in the design that are to use same-cycle or next-cycle checking, irrespective of any object-based settings and the global **timing_default_phasing** variable setting.

SEE ALSO

```
remove_no_same_phase_path(2)
remove_phasing(2)
remove_same_phase_path(2)
set_multicycle_path(2)
set_no_same_phase_path(2)
set_phasing(2)
set_same_phase_path(2)
timing_intersection_transparency(3)
```

timing_differential_iteration_count

Specifies the minimum number of path tracing iterations to be performed during differential delay skew analysis.

TYPE

integer

DEFAULT

1

DESCRIPTION

The **timing_differential_iteration_count** variable specifies the minimum number of timing iterations to perform for differential skew analysis. A value of 1 (the default) disables differential skew iteration. Any value greater than 1 enables the feature, and the number of iterations performed will equal or exceed the set value.

Differential iterations run concurrently with iterations for other features such as signal integrity or multi-input switching analysis. NanoTime updates the results for all enabled features during each path tracing iteration until all iteration exit criteria for the enabled features are satisfied. For this reason, the number of iterations might exceed the value set in the **timing_differential_iteration_count** variable.

SEE ALSO

`timing_multi_input_max_iteration(3)`
`si_xtalk_exit_on_max_iteration_count(3)`

timing_enable_multi_input_switching

Enables or disables timing-based multi-input switching analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime performs timing-based multi-input switching analysis. The default is **false**. Multi-input switching analysis requires a NanoTime Ultra license.

Timing-based multi-input switching analysis should not be used in conjunction with the **set_conservative_min_delay** or **set_conservative_max_delay** commands, which define user-guided multi-input switching analysis. The **set_conservative_min_delay** and **set_conservative_max_delay** commands should only be used if the **timing_enable_multi_input_switching** variable is set to **false**.

Timing-based multi-input switching analysis is an iterative analysis. The **timing_multi_input_exit_reevaluated_nets**, **timing_multi_input_exit_reevaluated_nets_pct**, and **timing_multi_input_max_iteration** variables control the iteration exit criteria.

The **timing_enable_multi_input_switching** variable must be set before executing the **check_design** command.

SEE ALSO

```
exclude_multi_input_switching(2)
report_multi_input_switching(2)
timing_multi_input_exit_reevaluated_nets(3)
timing_multi_input_exit_reevaluated_nets_pct(3)
timing_multi_input_max_iteration(3)
timing_multi_input_overlap_tolerance(3)
set_conservative_max_delay(2)
set_conservative_min_delay(2)
```

timing_enable_mux_xor_pessimism_removal

Specifies whether to search for and modify the delay calculation for 2-input mux-based XOR circuits.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

NanoTime automatically recognizes mux-based 2-input XOR and XNOR gates and treats them specially during delay calculation. This processing can be disabled by setting this variable to false.

A mux-based 2-input XOR is identified as two oppositely controlled tgates whose outputs are connected to the same net, and whose inputs are connected by an inverter.

The **check_design** command reports how many of these circuits are found, if any. The **check_design -message_level verbose** command reports which delay calculations are affected by this processing.

SEE ALSO

`check_design(2)`

timing_enable_non_transparent_setup_delta_d

Enables the addition of the delta delay from the related pin to the checked pin to the data arrival time when evaluating a non-transparent setup constraint.

TYPE

boolean

DEFAULT

true

DESCRIPTION

This global variable exists for backward compatibility with PathMill when NanoTime is used for custom model generation.

SEE ALSO

`extract_model(2)`

timing_enable_path_through_sense_amplifier

Determines whether NanoTime traces and reports paths through sense amplifier enable to memory read output.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime only trace paths through memory wordline select to memory read output. Set this variable to true to enable NanoTime trace and report paths through sense amplifier enable to memory read output. Such paths are typically seen in a pipelined memory design.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

timing_enable_skewed_mis_precharge_clock

Enables the use of offset input arrival times during simulation of parallel precharge clock devices that are being used simultaneously within one precharge topology.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

You can use this feature if a precharge topology has multiple, parallel precharge clock devices whose input signals are logically equivalent. The input nets connected to the precharge clock devices must be marked as merged with the **set_merged_nets** command. The offset skew between each input is computed and the input with the earliest arrival is used as a reference. Timing arcs from the other non-reference inputs are disabled. When simulating from the reference pin, the other parallel precharge devices are included with the appropriate input skew applied.

SEE ALSO

```
set_conservative_max_delay(2)
set_conservative_min_delay(2)
set_merged_nets(2)
timing_enable_multi_input_switching(3)
```

timing_enable_slew_variation

Enable transition based variation parametric on-chip variation (POCV) analysis. This requires the user to supply extra information when issuing the set_variation_parameters command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime uses the transition variation at the input of a stage when determining the overall variation (delay and output considered) of that stage. In this mode the variation of a stage is a function of the intrinsic variation of the stage and also a function of the transition variation at the input of the stage. For this to be effective, the user must specificy input_transitions, variations, and transition variations when specifying transistor variation values with the set_variation_parameters command. In this mode the transition variation values in a .lib (.db) cell will also be used.

The **timing_enable_slew_variation** variable must be set before executing any **set_variation_parameters** commands.

SEE ALSO

```
set_variation_parameters(2)
report_variation_parameters(2)
report_variation_calculation(2)
```

```
set_pin_variation(2)
report_pin_variation(2)
```

timing_exception_start_delay_max_fall

Specifies the data arrival time that is used at the startpoints when NanoTime traces the paths specified by one or more **set_find_path** commands, for maximum-delay falling transition path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This command specifies the delay value for falling transition at the startpoints, in nanoseconds. This value is used when NanoTime traces the paths specified by the **set_find_path** command.

SEE ALSO

```
set_find_path(2)
trace_paths(2)
timing_exception_start_delay_max_rise(3)
timing_exception_start_delay_min_rise(3)
timing_exception_start_delay_min_fall(3)
```

timing_exception_start_delay_max_rise

Specifies the data arrival time that is used at the startpoints when NanoTime traces the paths specified by one or more **set_find_path** commands, for maximum-delay rising transition path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This command specifies the delay value for rising transition at the startpoints, in nanoseconds. This value is used when NanoTime traces the paths specified by the **set_find_path** command.

SEE ALSO

```
set_find_path(2)
trace_paths(2)
timing_exception_start_delay_min_rise(3)
timing_exception_start_delay_min_fall(3)
timing_exception_start_delay_max_fall(3)
```

timing_exception_start_delay_min_fall

Specifies the data arrival time that is used at the startpoints when NanoTime traces the paths specified by one or more **set_find_path** commands, for minimum-delay falling transition path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This command specifies the delay value for falling transition at the startpoints, in nanoseconds. This value is used when NanoTime traces the paths specified by the **set_find_path** command.

SEE ALSO

```
set_find_path(2)
trace_paths(2)
timing_exception_start_delay_max_rise(3)
timing_exception_start_delay_max_fall(3)
timing_exception_start_delay_min_rise(3)
```

timing_exception_start_delay_min_rise

Specifies the data arrival time that is used at the startpoints when NanoTime traces the paths specified by one or more **set_find_path** commands, for minimum-delay rising transition path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This command specifies the delay value for rising transition at the startpoints, in nanoseconds. This value is used when NanoTime traces the paths specified by the **set_find_path** command.

SEE ALSO

```
set_find_path(2)
trace_paths(2)
timing_exception_start_delay_max_rise(3)
timing_exception_start_delay_max_fall(3)
timing_exception_start_delay_min_fall(3)
```

timing_extended_sidebranch_analysis_level

Specifies the default effort level for the side branch exploration algorithm.

TYPE

integer

DEFAULT

0

DESCRIPTION

The **timing_extended_sidebranch_analysis_level** variable specifies the global effort level to use for sidebranch exploration. Valid values are 0, 1, 2, or 3.

By default, during the computation of stage delay, simulation units with a large number of inputs are pruned to use devices only in the switching path to power or ground. This approach is partial exploration of the stage. While this approach significantly improves the performance and capacity of the tool, it might not provide the desired level of pessimism in certain situations. Full exploration of the side branches of a stage can add more pessimism.

Set the **timing_extended_sidebranch_analysis_level** variable to adjust the global effort level.

Each increment in the value increases the effort level, which widens the set of stages for which side branches are fully explored.

Using a high global effort level might lead to a significant increase in runtime and memory. You can override the global effort level for specific nets by using the **set_extended_sidebranch_level** command. Net-specific settings override the global setting.

SEE ALSO

set_extended_sidebranch_level(2)

timing_flip_flop_hold_fall_margin

Specifies the amount of timing margin for falling-edge hold checks at flip-flops.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge hold checks at flip-flops. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any hold check at a flip-flop.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_flip_flop_hold_rise_margin(3)
timing_flip_flop_setup_fall_margin(3)
timing_flip_flop_setup_rise_margin(3)
```

timing_flip_flop_hold_rise_margin

Specifies the amount of timing margin for rising-edge hold checks at flip-flops.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge hold checks at flip-flops. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any hold check at a flip-flop.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_flip_flop_hold_fall_margin(3)
timing_flip_flop_setup_fall_margin(3)
timing_flip_flop_setup_rise_margin(3)
```

timing_flip_flop_setup_fall_margin

Specifies the amount of timing margin for falling-edge setup checks at flip-flops.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge setup checks at flip-flops. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any setup check at a flip-flop.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_flip_flop_hold_rise_margin(3)
timing_flip_flop_hold_fall_margin(3)
timing_flip_flop_setup_rise_margin(3)
```

timing_flip_flop_setup_rise_margin

Specifies the amount of timing margin for rising-edge setup checks at flip-flops.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge setup checks at flip-flops. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any setup check at a flip-flop.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_flip_flop_hold_rise_margin(3)
timing_flip_flop_hold_fall_margin(3)
timing_flip_flop_setup_fall_margin(3)
```

timing_high_impedance_pessimism_reduction

Enables or disables the consideration of pessimism reduction due to high impedance net during delay calculation

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime assigns a pessimistic initial condition to the high impedance net during delay calculation. When this variable is set to **true**, NanoTime uses an initial condition that matches the final voltage after switching.

EXAMPLE

An example of high impedance net in delay calculation is the long channel inverter, when input is rising, the intermediate net in **NMOS** stack is in high impedance state. When this variable is **false**, the initial condition of the intermediate net in **NMOS** stack for max tracing is set to **weak VDD**. when this variable is **true**, the initial condition is set to ground, which equals to **0**.

timing_intersection_transparency

Sets same-phase checking for all launch and capture clocks that have overlapping on times.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If the **timing_intersection_transparency** variable is **true**, NanoTime uses same-phase checking when there is any amount of overlap between the transparency windows of clocks driving the launch and capture latches. An overlap occurs when the opening edge of the launch clock occurs before the closing edge of the capture clock. The amount of overlap must be greater than zero to cause same-phase checking.

When the variable is **false** (the default), NanoTime performs same-phase checking only when the opening edges at the launch and capture latches occur at the same time.

For more information about same-phase and no-same-phase checking, see the man page for the **set_phasing** command.

SEE ALSO

```
remove_phasing(2)
remove_no_same_phase_path(2)
```

```
remove_same_phase_path(2)
report_exceptions(2)
set_phasing(2)
set_no_same_phase_path(2)
set_same_phase_path(2)
```

timing_latch_hold_fall_margin

Specifies the amount of timing margin for falling-edge hold checks at latches.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge hold checks at latches. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any hold check at a latch.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_latch_hold_rise_margin(3)
timing_latch_setup_fall_margin(3)
timing_latch_setup_rise_margin(3)
```

timing_latch_hold_rise_margin

Specifies the amount of timing margin for rising-edge hold checks at latches.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge hold checks at latches. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any hold check at a latch.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_latch_hold_fall_margin(3)
timing_latch_setup_fall_margin(3)
timing_latch_setup_rise_margin(3)
```

timing_latch_setup_fall_margin

Specifies the amount of timing margin for falling-edge setup checks at latches.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge setup checks at latches. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any setup check at a latch.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_latch_hold_rise_margin(3)
timing_latch_hold_fall_margin(3)
timing_latch_setup_rise_margin(3)
```

timing_latch_setup_rise_margin

Specifies the amount of timing margin for rising-edge setup checks at latches.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge setup checks at latches. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any setup check at a latch.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_latch_hold_rise_margin(3)
timing_latch_hold_fall_margin(3)
timing_latch_setup_fall_margin(3)
```

timing_max_delay_paths_saved

Specifies the largest number of maximum-delay paths saved during path tracing.

TYPE

integer

DEFAULT

100000000

DESCRIPTION

This variable specifies the largest number of maximum-delay paths that can be saved into the path database. If the specified limit is reached during path tracing, NanoTime discards saved paths that have smaller delay values and saves new paths that have larger delay values.

Set this number lower to reduce memory usage or to limit the size of reports generated by the **report_paths** command. Set it larger to keep more worst-case paths in the database.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_max_slack_paths_saved(3)
timing_min_delay_paths_saved(3)
timing_min_slack_paths_saved(3)
```

timing_max_path_transparency_begin_margin

Specifies the amount of timing margin applied to the opening of transparency windows at sequential elements during maximum-delay checking.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable controls the amount of margin applied to maximum-delay timing constraints that check for the opening of the transparency window at sequential elements such as latches, precharge structures, and register files.

By default, NanoTime assumes that the data can arrive at a transparent sequential element just as the window opens, or at any time after the window opens, for propagation of the data to continue. Specifying a positive margin requires the data to arrive after the window opens by at least the specified amount of time. On the other hand, specifying a negative value allows the data to arrive before the window opens by the specified amount of time.

This variable affects the behavior of path tracing. A positive value results in more restrictive checking, causing the path tracer to trace through fewer transparent devices. This results in shorter paths and fewer timing checks. A negative value has the opposite effect.

This variable does not directly affect reported slack values.

SEE ALSO

```
trace_paths(2)
timing_min_path_transparency_begin_margin(3)
```

timing_max_slack_paths_saved

Specifies the largest number of least-slack, maximum-delay paths that can be saved during path tracing.

TYPE

integer

DEFAULT

100000000

DESCRIPTION

This variable specifies the largest number of maximum-delay paths having the least (most negative) timing slack that can be saved into the path database. If the specified limit is reached during path tracing, NanoTime discards saved paths that have larger slack values and saves new paths that have smaller slack values.

Set this number lower to reduce memory usage or to limit the size of reports generated by the **report_paths** command. Set it larger to keep more worst-case paths in the database.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_max_delay_paths_saved(3)
timing_min_delay_paths_saved(3)
timing_min_slack_paths_saved(3)
```

timing_memory_bitline_valid_before_wordline

Adds an additional check requiring that the bitline signal should be valid before the wordline control signal is turned on during write mode.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime for memories creates a setup timing check that requires the data into a bitcell to be valid before the wordline signal controlling the input into that bitcell is turned off. A more conservative check can be added that requires that the bitline signal at the input into the wordline controlled device be valid before the wordline device is turned on. This setup check will be type M16, with a label of "bit line valid before word line on". Set this variable to true to enable this check. This can cause additional runtime during write mode memory analysis.

timing_memory_measurement_bitline_precharg

Specifies the voltage level needed to precharge the memory bitlines to during the precharge phase. A 0.0 value indicates no precharge checking will take place.

TYPE

float

DEFAULT

0.0

DESCRIPTION

During the operation of a memory, there is precharge operation that occurs between each read or write operation. This precharge operation charges the bitlines to a "high" voltage level during the precharge phase so that the subsequent operation can be performed correctly. By default, NanoTime does not check the timing for this precharge phase. To enable precharge phase checking, set this variable to a non-zero value between 0 and 100 that indicates the percentage of the bitline supply voltage which is high enough to be precharged. This will cause the addition of an M17 type timing check, performing a setup check requiring the time it takes for the bitline to achieve the user specified voltage before the time when the precharge control signal turns off.

SEE ALSO

`timing_memory_measurement_differential_voltage_percentage(3)`
`timing_memory_measurement_sense_amp_trigger_threshold_percentage(3)`

timing_memory_measurement_differential_volt

Specifies a custom differential voltage check value as a percentage of the supply voltage for the sense amplifier input differential voltage measurement.

TYPE

float

DEFAULT

0.1

DESCRIPTION

Set the **timing_memory_measurement_differential_voltage_percentage** variable to specify a custom differential voltage check value as a percentage value of the supply for the sense amplifier input measurement executed by the **report_measurement** command. When the sense amplifier enable trigger transition crosses the threshold value specified by the **timing_memory_measurement_sense_amp_trigger_threshold_percentage** variable, the differential voltage is measured on the two inputs of the sense amplifier and compared against the value set by the **timing_memory_measurement_differential_voltage_percentage** variable. If the measured differential voltage is smaller than the value set by this variable, the measurement is reported as a violator by the **report_measurement** command.

SEE ALSO

```
report_measurement(2)
mark_sense_amp(2)
erase_sense_amp(2)
```

```
timing_memory_measurement_sense_amp_trigger_threshold_percentage(3)
```

timing_memory_measurement_sense_amp_trig

Specifies the trigger threshold of the sense amplifier enable signal (as a percentage of the supply voltage) to define when the **report_measurement** command reports the sense amplifier input differential voltage.

TYPE

float

DEFAULT

0.5

DESCRIPTION

The **report_measurement** command reports the differential voltage measurement on the sense amplifier inputs. Set the **timing_memory_measurement_sense_amp_trigger_threshold_percentage** variable to specify the voltage on the trigger of the sense amplifier enable signal at which the measurement is taken. The argument is specified as a percentage of the supply voltage.

When the sense amp enable trigger transition crosses this level, the differential voltage is measured on the two inputs of the sense amplifier. The **report_measurement** command is used to report all such measurements for the current analysis.

SEE ALSO

```
report_measurement(2)
mark_sense_amp(2)
erase_sense_amp(2)
```

```
timing_memory_measurement_differential_voltage_percentage(3)
```

timing_memory_read_mux_before_wordline

Changes the required order of signal arrivals for a setup check between the read mux control and the wordline control.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime for memories creates a setup timing check that requires the wordline control to be valid before the read mux control. To reverse the required order of these two signals, set the **timing_memory_read_mux_before_wordline** variable to **true**. The setup check then requires the read mux signal to be valid before the wordline control.

SEE ALSO

`timing_memory_remove_sense_amp_before_read_mux(3)`

timing_memory_remove_sense_amp_before_re

Removes the timing check requirement that the sense amp enable pin is valid before the read mux control signal.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime for memories creates a setup timing check that requires the sense amp enable signal to be valid before the read mux control signal.

If you set the **timing_memory_remove_sense_amp_before_read_mux** variable to **true**, the timing check is not created.

SEE ALSO

`timing_memory_read_mux_before_wordline(3)`

timing_memory_simultaneous_read

Takes simultaneous reading of both ports in a dual-port memory design into account during delay analysis.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Dual-port memory designs might have the capability of reading the same address from both ports during a single operation. When this occurs, the access time of the read operation might increase. NanoTime models this additional delay if the **timing_memory_simultaneous_read** variable is set to **true**. If the variable is set to **false**, only one port can read from a single address at a time.

SEE ALSO

timing_memory_use_preswitched_read_mux

Allows the read mux control signal to remain in a fixed on state during analysis of a memory bit column.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime for memories assumes that the input signals to a memory bit column are switching in a correlated manner, typically coming from paths originated by a common clock edge. The read mux control signal is one of these signals and it is assumed to be turning on during the simulation.

In some cases, the read mux control signal is already stable before the wordline or sense amp enable signals switch. In this situation the read mux control should be in the on state and not switching when the bit column is analyzed. For this case, set the **timing_memory_use_preswitched_read_mux** variable to **true**.

SEE ALSO

timing_merge_parallel_transistors

Specifies whether to merge parallel transistors for path tracing.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime merges parallel transistors for purposes of path tracing. Two or more transistors are parallel if they are the same type of transistor (NMOS or PMOS) and have the same source, gate, and drain pin connections.

If you do not want NanoTime to merge parallel transistors for path tracing, set the **timing_merge_parallel_transistors** variable to **false**. In that case, the timing paths through the parallel transistors are maintained and calculated separately, resulting in more timing paths. You might want to do this to get a more accurate analysis of parasitics between the parallel transistors.

The **timing_merge_parallel_transistors** variable affects the merging done for path tracing only. The **pattern_merge_parallel_transistors** variable affects the merging done for pattern recognition.

SEE ALSO

`trace_paths(2)`
`pattern_merge_parallel_transistors(3)`

timing_min_delay_paths_saved

Specifies the largest number of minimum-delay paths saved during path tracing.

TYPE

integer

DEFAULT

100000000

DESCRIPTION

This variable specifies the largest number of minimum-delay paths that can be saved into the path database. If the specified limit is reached during path tracing, NanoTime discards saved paths that have larger delay values and saves new paths that have smaller delay values.

Set this number lower to reduce memory usage or to limit the size of reports generated by the **report_paths** command. Set it larger to keep more worst-case paths in the database.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_max_delay_paths_saved(3)
timing_max_slack_paths_saved(3)
timing_min_slack_paths_saved(3)
```

timing_min_iteration_count

Specifies the minimum number of iterations to be performed during SI/MIS/differential delay skew analysis.

TYPE

integer

DEFAULT

3

DESCRIPTION

The **timing_min_iteration_count** variable specifies the minimum number of iterations to perform during SI/MIS/differential skew analysis. A value of 1 disables the impact of this variable. The default value is set to 3.

When a value N is set to **timing_min_iteration_count** Nanotime will optimize the runtime in the first N-1 iterations. If a value larger than the maximum number of iterations is set the variable will be bounded by the maximum number of iterations.

SEE ALSO

```
timing_differential_iteration_count(3)
timing_multi_input_max_iteration(3)
si_xtalk_exit_on_max_iteration_count(3)
```

timing_min_path_transparency_begin_margin

Specifies the amount of timing margin applied to the opening of transparency windows at sequential elements during minimum-delay checking.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable controls the amount of margin applied to minimum-delay timing constraints that check for the opening of the transparency window at sequential elements such as latches, precharge structures, and register files.

By default, NanoTime assumes that the data can arrive at a transparent sequential element just as the window opens, or at any time after the window opens, for propagation of the data to continue. Specifying a positive margin requires the data to arrive after the window opens by at least the specified amount of time. On the other hand, specifying a negative value allows the data to arrive before the window opens by the specified amount of time.

This variable affects the behavior of path tracing. A positive value results in more restrictive checking, causing the path tracer to trace through fewer transparent devices. This results in shorter paths and fewer timing checks. A negative value has the opposite effect.

This variable does not directly affect reported slack values.

SEE ALSO

trace_paths(2)
timing_max_path_transparency_begin_margin(3)

timing_min_slack_paths_saved

Specifies the largest number of least-slack, minimum-delay paths that can be saved during path tracing.

TYPE

integer

DEFAULT

100000000

DESCRIPTION

This variable specifies the largest number of minimum-delay paths having the least (most negative) timing slack that can be saved into the path database. If the specified limit is reached during path tracing, NanoTime discards saved paths that have larger slack values and saves new paths that have smaller slack values.

Set this number lower to reduce memory usage or to limit the size of reports generated by the **report_paths** command. Set it larger to keep more worst-case paths in the database.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_max_delay_paths_saved(3)
timing_max_slack_paths_saved(3)
timing_min_delay_paths_saved(3)
```

timing_multi_input_exit_reevaluated_nets

Specifies the maximum number of nets selected for reevaluation during timing-based multi-input switching analysis.

TYPE

integer

DEFAULT

0

DESCRIPTION

Specifies the maximum number of nets selected for reevaluation when timing-based multi-input switching analysis is enabled by setting the **timing_enable_multi_input_switching** variable to **true**. Multi-input switching analysis requires a NanoTime Ultra license.

Timing-based multi-input switching analysis is an iterative analysis. NanoTime exits the analysis loop after completing the current iteration when the number of nets selected for reevaluation in the next iteration is less than the value specified in the **timing_multi_input_exit_reevaluated_nets** variable.

This variable is one of several variables that determine iteration exit criteria. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **timing_multi_input_max_iteration** variable.
2. The number of nets selected for reevaluation in the next iteration is less than the value of the **timing_multi_input_exit_reevaluated_nets** variable.
3. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **timing_multi_input_exit_reevaluated_nets_pct** variable.

4. You manually exit the analysis loop by pressing Ctrl+C to send an interrupt signal to the NanoTime process. You cannot interrupt an iteration without exiting NanoTime.

SEE ALSO

```
timing_enable_multi_input_switching(3)
exclude_multi_input_switching(2)
report_multi_input_switching(2)
timing_multi_input_exit_reevaluated_nets_pct(3)
timing_multi_input_max_iteration(3)
timing_multi_input_overlap_tolerance(3)
```

timing_multi_input_exit_reevaluated_nets_pct

Specifies the maximum percentage of nets selected for reevaluation relative to the total number of nets, below which NanoTime exits the multi-input switching analysis loop.

TYPE

float

DEFAULT

0

DESCRIPTION

Specifies a maximum percentage of nets selected for reevaluation relative to the total number of nets when timing-based multi-input switching analysis is enabled by setting the **timing_enable_multi_input_switching** variable to **true**. Multi-input switching analysis requires a NanoTime Ultra license.

Timing-based multi-input switching analysis is an iterative analysis. NanoTime exits the analysis loop after completing the current iteration when the percentage of nets selected for reevaluation in the next iteration is less than value specified in the **timing_multi_input_exit_reevaluated_nets_pct** variable. The number of nets includes all individual net segments in the same way that the **[get_nets -hierarchical *]** command counts all nets in the design.

This variable is one of several variables that determine iteration exit criteria. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **timing_multi_input_max_iteration** variable.
2. The number of nets selected for reevaluation in the next iteration is less than the value of the **timing_multi_input_exit_reevaluated_nets** variable.

3. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **timing_multi_input_exit_reevaluated_nets_pct** variable.
4. You manually exit the analysis loop by pressing Ctrl+C to send an interrupt signal to the NanoTime process. You cannot interrupt an iteration without exiting NanoTime.

SEE ALSO

```
timing_enable_multi_input_switching(3)
exclude_multi_input_switching(2)
report_multi_input_switching(2)
timing_multi_input_max_iteration(3)
timing_multi_input_exit_reevaluated_nets(3)
timing_multi_input_overlap_tolerance(3)
```

timing_multi_input_max_iteration

Specifies the maximum number of incremental timing iterations during timing-based multi-input switching analysis.

TYPE

integer

DEFAULT

3

DESCRIPTION

Specifies the maximum number of incremental timing iterations when timing-based multi-input switching analysis is enabled by setting the **timing_enable_multi_input_switching** variable to **true**. Multi-input switching analysis requires a NanoTime Ultra license.

Timing-based multi-input switching analysis is an iterative analysis. NanoTime exits the analysis loop after performing the number of iterations specified in this variable. The default is 3, meaning that NanoTime exits the analysis loop after performing three iterations. The minimum allowed value is 1.

This variable is one of several variables that determine iteration exit criteria. NanoTime exits the analysis loop after completing the current iteration if one or more of the following is true:

1. The number of iterations performed equals the value of the **timing_multi_input_max_iteration** variable.
2. The number of nets selected for reevaluation in the next iteration is less than the value of the **timing_multi_input_exit_reevaluated_nets** variable.
3. The percentage of nets (relative to the total number of nets) selected for reevaluation is less than the value of the **timing_multi_input_exit_reevaluated_nets_pct** variable.

4. You manually exit the analysis loop by pressing Ctrl+C to send an interrupt signal to the NanoTime process. You cannot interrupt an iteration without exiting NanoTime.

SEE ALSO

```
timing_enable_multi_input_switching(3)
exclude_multi_input_switching(2)
report_multi_input_switching(2)
timing_multi_input_exit_reevaluated_nets(3)
timing_multi_input_exit_reevaluated_nets_pct(3)
timing_multi_input_overlap_tolerance(3)
```

timing_multi_input_overlap_tolerance

Specifies the overlap (in nanoseconds) for signals to be considered when timing-based multi-input switching analysis is enabled by setting the **timing_enable_multi_input_switching** variable to **true**. Multi-input switching analysis requires a NanoTime Ultra license.

The **timing_multi_input_overlap_tolerance** variable sets the tolerance to determine if two or more input windows overlap. Setting this value to a large number might produce unstable results.

TYPE

float

DEFAULT

0.003

DESCRIPTION

This variable specifies a tolerance that qualifies two or more inputs of a channel-connected region for timing-based multi-input switching analysis.

SEE ALSO

```
timing_enable_multi_input_switching(3)
exclude_multi_input_switching(2)
report_multi_input_switching(2)
timing_multi_input_exit_reevaluated_nets(3)
timing_multi_input_exit_reevaluated_nets_pct(3)
timing_multi_input_max_iteration(3)
```

timing_pbsa_gate_delay_only

Determines whether to apply path-based slack adjustment factors to the gate delay only or to the combined gate and wire delay portions of each arc delay.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Path-based slack adjustment (PBSA) is a method to modify path delays by an amount proportional to the length of each path segment. You must have a NanoTime Ultra license to use path-based slack adjustment.

By default, NanoTime applies PBSA scaling factor variables such as the **pbsa_KAmax**, **pbsa_KALmax**, and **pbsa_KALpctmax** variables to the total arc delay, which includes the gate delay and the wire delay.

If you set the **timing_pbsa_gate_delay_only** variable to **true**, the gate delay and wire delay portions of an arc delay are scaled independently. The **pbsa_*** variables that do not include "wire" in their names are applied to the gate delay portion only. Variables of the form **pbsa_*** that include "wire" in their names (for example, the **pbsa_KAwiremax**, **pbsa_KAwiremin**, and **pbsa_KBwiremax** variables) are applied to the wire delay portion on each path segment. The scaled gate delay and scaled wire delay are added to obtain the total arc delay.

Setting the **timing_pbsa_gate_delay_only** variable to **true** automatically sets the **timing_save_wire_delay** variable to **true** to separate the arc delays into gate and wire delays.

For more information, see the NanoTime User guide. Using this feature might cause a significant increase in memory usage and possibly runtime.

SEE ALSO

timing_save_wire_delay(3)
report_pbsa_calculation(2)
pbsa_KAwiremax(3)
pbsa_KAwiremin(3)
pbsa_KBwiremax(3)
pbsa_KBwiremin(3)
pbsa_KCwiremax(3)
pbsa_KCwiremin(3)
pbsa_KDwiremax(3)
pbsa_KDwiremin(3)

timing_pocv_gate_delay_only

Specifies if RC delays should be included when computing the variation of a stage.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

To calculate the variation of a path, the NanoTime tool first computes the variation of each stage. When computing the delay variation of a stage, the tool applies a factor to a delay value. The delay used can either be just the gate delay of the stage, or the combined gate delay and RC delay of that stage. By default, only the gate delay is used to compute the variation of each stage. Setting the **timing_pocv_gate_delay_only** variable to **false** includes the RC delay in this calculation.

SEE ALSO

```
trace_paths(2)
set_variation_parameters(2)
set_libcell_variation_parameters(2)
```

timing_pocv_sigma

Specifies the number of standard deviations to use when calculating variation for path delays.

TYPE

float

DEFAULT

3

DESCRIPTION

The **set_variation_parameters** and **set_libcell_variation_parameters** specify variation for individual arc delays. The values specified by these commands are defined as a one-sigma variation. You can specify the level of variation to account for when computing the path delay and subsequent slack by modifying the **timing_pocv_sigma** variable.

The default is a three sigma variation. For a less conservative approach, set the **timing_pocv_sigma** variable to a value less than three. The effects of variation calculations appear in slack calculations and path reporting. This value will apply to both min and max paths unless timing_pocv_sigma_min is set to a value >= 0.0. In that case the timing_pocv_sigma_min value will be used for min checked paths.

SEE ALSO

```
timing_pocv_sigma_min(3)
set_variation_parameters(2)
set_libcell_variation_parameters(2)
report_variation(2)
```

timing_pocv_sigma_min

Specifies the number of standard deviations to use when calculating variation for path delays of min checked paths.

TYPE

float

DEFAULT

-1

DESCRIPTION

The **set_variation_parameters** and **set_libcell_variation_parameters** specify variation for individual arc delays. The values specified by these commands are defined as a one-sigma variation. You can specify the level of variation to account for when computing the path delay and subsequent slack by modifying the **timing_pocv_sigma** and/or the **timing_pocv_sigma_min** variable.

The default value for timing_pocv_sigma_min is -1, which indicates that the value from the timing_pocv_sigma variable will be used for both min and max paths. If this variable is set to a value >= 0 then all min checked paths will use this sigma value for computing variation on a min checked path for both the checked and corresponding reference path.

SEE ALSO

```
timing_pocv_sigma(3)
set_variation_parameters(2)
set_libcell_variation_parameters(2)
```

```
report_variation(2)
```

timing_port_capacitance_net_depth_limit

Specifies a maximum depth level of nets from a port when traversing pass transistors to compute capacitance loading.

TYPE

integer

DEFAULT

2

DESCRIPTION

This variable sets the upper bound of the depth level of nets from a port when NanoTime traverses pass transistors based on their logic states to calculate capacitance loading.

SEE ALSO

timing_precharge_check_contention_between_

Enables checking for contention between the precharge and evaluate clocks of D1 type domino circuits when different clocks are used for the precharge and evaluate phases.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

A D1 type domino circuit has a controlling clock on the evaluate stack. This clock keeps the evaluate stack from conducting during the precharge phase.

By default, NanoTime checks for contention between the precharge and evaluate clocks of D1 type domino circuits when different clocks are used for the precharge and evaluate phases. This clock-to-clock check is performed after path tracing. To view the results of the check, use the **report_constraint** command with the **-clock_setup** and **-clock_hold** options.

To disable this type of checking, set the **timing_precharge_check_contention_between_pchg_eval_clocks** variable to **false**.

When this type of checking is enabled, NanoTime verifies that the amount of time between the precharge and evaluate clock transitions is at least zero. To specify a different time constraint, set the **timing_precharge_pchg_eval_clocks_contention_margin** variable to a negative value for a more restrictive check, or to a positive value for a looser check.

A separate variable controls the checking of D1 type domino circuits when the *same* clock is used for both the precharge and evaluate phases. This type of contention checking is disabled by default.

SEE ALSO

report_constraint(2)
trace_paths(2)
timing_precharge_check_contention_between_same_pchg_eval_clock(3)
timing_precharge_pchg_eval_clocks_contention_margin(3)
timing_precharge_same_pchg_eval_clock_contention_margin(3)

timing_precharge_check_contention_between_s

Enables checking for contention between the precharge and evaluate clock transitions of D1 type domino circuits when the same clock is used for both the precharge and evaluate phases.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

A D1 type domino circuit has a controlling clock on the evaluate stack. This clock keeps the evaluate stack from conducting during the precharge phase.

By default, NanoTime does not check for contention between the precharge and evaluate clock transitions of D1 type domino circuits when the same clock is used for both the precharge and evaluate phases. To enable this type of checking, set the **timing_precharge_check_contention_between_same_pchg_eval_clock** variable to **true**. This check is performed after path tracing. To view the results of the check, use the **report_constraint** command with the **-clock_setup** and **-clock_hold** options.

When this type of checking is enabled, NanoTime verifies that the amount of time between the precharge and evaluate clock transitions is at least zero. To specify a different time constraint, set the **timing_precharge_same_pchg_eval_clock_contention_margin** variable to a negative value for a more restrictive check, or to a positive value for a looser check.

A separate variable controls the checking of D1 type domino circuits when *different* clocks are used for the precharge and evaluate phases. This type of contention checking is enabled by default.

SEE ALSO

report_constraint(2)
trace_paths(2)
timing_precharge_check_contention_between_pchg_eval_clocks(3)
timing_precharge_pchg_eval_clocks_contention_margin(3)
timing_precharge_same_pchg_eval_clock_contention_margin(3)

timing_precharge_contention_check

Specifies whether to perform contention checking at D2 type domino stages during the precharge phase.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable specifies whether to perform contention checking at D2 type domino stages during the precharge phase. A D2 stage does not have a controlling clock on the evaluation stack, so the evaluation stack should be turned off during the precharge phase.

If the variable is set to **true** (the default), NanoTime performs type S2a setup checking and type H3a hold checking, which check input signals arrivals with respect to the precharge phase. If the variable is set to **false**, NanoTime performs type S2b setup checking and type H3b hold checking, which check input signal arrivals with respect to the evaluate phase.

When the variable is set to **true**, for a falling input signal, NanoTime verifies that the falling edge occurs before the start of the precharge phase. This is called the S2a setup check. NanoTime also verifies that the signal remains low at least until the end of the precharge phase, to prevent discharge of the precharge node. This is called the H3a hold check.

When this variable is **false**, NanoTime verifies that the falling edge of the input signal arrives soon enough before the start of the evaluate phase to prevent unwanted discharge during the evaluate phase. This is called the S2b setup check. NanoTime also verifies that the signal remains low at least until the end of the evaluate phase, to prevent incorrect discharge during that phase. This is called the H3b hold check.

SEE ALSO

trace_paths(2)
timing_precharge_contention_recovery(3)

timing_precharge_contention_recovery

Controls the handling of contention violations at D2 type domino stages.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls the handling contention at D2 type domino stages. A D2 stage does not have a controlling clock on the evaluation stack, so the evaluation stack should be turned off during the precharge phase.

During a maximum-delay path searching, if a precharge input to a D2 stage changes from low to high during the precharge cycle, it can cause contention, depending on the setup margin that has been defined. If there is no contention violation, NanoTime adjusts and shifts the arrival time of the input signal to the D2 domino node to the end of the precharge cycle, and then continues path tracing.

If there is a contention violation, the **timing_precharge_contention_recovery** variable determines the action taken. If the variable is **false** (the default), NanoTime stops path tracing and does not consider the path. If the variable is **true**, NanoTime shifts and adjusts the arrival time to the end of the precharge cycle and continues path tracing.

SEE ALSO

```
trace_paths(2)
```

timing_precharge_hold_fall_margin

Specifies the amount of timing margin for falling-edge hold checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge hold checks at the constrained (data) pin of precharge structures. The specified margin value applies to a falling-edge signal arriving at the constrained pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraints** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_hold_fall_margin(3)
timing_flip_flop_hold_fall_margin(3)
timing_latch_hold_fall_margin(3)
timing_precharge_in_hold_fall_margin(3)
timing_ram_hold_fall_margin(3)
```

timing_precharge_hold_rise_margin

Specifies the amount of timing margin for rising-edge hold checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge hold checks at the constrained (data) pin of precharge structures. The specified margin value applies to a rising-edge signal arriving at the constrained pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraints** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_hold_rise_margin(3)
timing_flip_flop_hold_rise_margin(3)
timing_latch_hold_rise_margin(3)
timing_precharge_in_hold_rise_margin(3)
timing_ram_hold_rise_margin(3)
```

timing_precharge_in_hold_fall_margin

Specifies the amount of timing margin for falling-edge hold checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge hold checks at the inputs of precharge structures. The specified margin value applies to a falling-edge signal arriving at the input pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraints** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_hold_fall_margin(3)
timing_flip_flop_hold_fall_margin(3)
timing_latch_hold_fall_margin(3)
timing_precharge_hold_fall_margin(3)
timing_ram_hold_fall_margin(3)
```

timing_precharge_in_hold_rise_margin

Specifies the amount of timing margin for rising-edge hold checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge hold checks at the inputs of precharge structures. The specified margin value applies to a rising-edge signal arriving at the input pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraints** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_hold_rise_margin(3)
timing_flip_flop_hold_rise_margin(3)
timing_latch_hold_rise_margin(3)
timing_precharge_hold_rise_margin(3)
timing_ram_hold_rise_margin(3)
```

timing_precharge_in_setup_fall_margin

Specifies the amount of timing margin for falling-edge setup checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge setup checks at the inputs of precharge structures. The specified margin value applies to a falling-edge signal arriving at the input pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraints** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_setup_fall_margin(3)
timing_flip_flop_setup_fall_margin(3)
timing_latch_setup_fall_margin(3)
timing_precharge_setup_fall_margin(3)
timing_ram_setup_fall_margin(3)
```

timing_precharge_in_setup_rise_margin

Specifies the amount of timing margin for rising-edge setup checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge setup checks at the inputs of precharge structures. The specified margin value applies to a rising-edge signal arriving at the input pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraints** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_setup_rise_margin(3)
timing_flip_flop_setup_rise_margin(3)
timing_latch_setup_rise_margin(3)
timing_precharge_setup_rise_margin(3)
timing_ram_setup_rise_margin(3)
```

timing_precharge_min_contention_recovery

Controls the handling of contention during minimum path tracing at a D2 type domino stage.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls the handling of contention violations at D2 type domino stages during minimum-delay path tracing. A D2 stage does not have a controlling clock on the evaluation stack, so the evaluation stack should be turned off during the precharge phase.

SEE ALSO

`trace_paths(2)`

timing_precharge_min_error_recovery

Allows tracing of a path to continue when a failing precharge hold check occurs.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, for a failing hold check on a precharge input that is turning on, NanoTime adjusts the arrival time to the earliest possible nonviolating arrival time and continues tracing the path forward. If this variable is set to **false**, NanoTime stops tracing a path when it detects a failing precharge hold check.

SEE ALSO

`trace_paths(2)`

timing_precharge_min_latch_transparency

Stops hold checking for data turning on for any data input of a precharge gate.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime performs a hold check for data turning on for any data input of a precharge gate. For an N-type precharge gate, this would be data rising versus evaluate clock falling. For a P-type precharge (or predischarge) gate, this would be data falling versus evaluate clock rising.

Also, NanoTime stops tracing a short path from a precharge data input at the evaluate node of the precharge gate.

Setting this variable to **true** tells NanoTime not to perform this hold check and to propagate the transition forward, for any precharge gate which is not classified as a domino flip-flop.

SEE ALSO

`timing_precharge_min_transparency(3)`

timing_precharge_min_transparency

Stops hold checking for data turning on versus the evaluate clock turning off for any data input of a precharge gate.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime performs a hold check for data turning on versus evaluate clock turning off for any data input of a precharge gate. For an N-type precharge gate, this would be data rising versus evaluate clock falling. For a P-type precharge (or predischarge) gate, this would be data falling versus evaluate clock rising.

Also, NanoTime stops tracing a short path from a precharge data input at the evaluate node of the precharge gate.

Setting this variable to **true** tells NanoTime not to perform this hold check and to propagate the transition forward, for any precharge gate which is not classified as a domino flip-flop.

SEE ALSO

`timing_precharge_min_latch_transparency(3)`

timing_precharge_other_create_delay_arcs

Enables the creation of delay arcs to intermediate nets in the evaluation stack of a precharge structure.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to true, enables the creation of delay arcs to intermediate nets in the evaluation stack of a precharge structure when the nets have "precharge other" devices connected to them. This allows user-defined timing checks installed at those nets to be evaluated during path tracing.

SEE ALSO

`create_timing_check(2)`

timing_precharge_pchg_eval_clocks_contentior

Sets the minimum amount of time between the precharge and evaluate clock transitions for clock-to-clock checking.

TYPE

float

DEFAULT

0

DESCRIPTION

When the **timing_precharge_check_contention_between_same_pchg_eval_clock** variable is set to **true**, NanoTime checks for contention between the precharge and evaluate clocks of D1 type domino circuits.

By default, when this type of checking is enabled, the amount of time between the precharge and evaluate clock transitions must be at least zero. To specify a different time constraint, set the **timing_precharge_pchg_eval_clocks_contention_margin** variable to a nonzero value. The specified value is the minimum allowable time separation between the precharge and evaluate clock transitions, in nanoseconds.

Specify a positive value for a more conservative check that requires more time separation, to ensure that contention does not occur. Specify a negative value for a more relaxed check that allows some overlap between the precharge and evaluate phases.

SEE ALSO

```
trace_paths(2)
timing_precharge_check_contention_between_same_pchg_eval_clock(3)
```

timing_precharge_prop_precharge_time

Controls path tracing from the precharge clock pin to the evaluate net of domino precharge circuits.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime traces paths from the precharge clock pin to the evaluate net. Setting this variable to **false** causes NanoTime not to trace these paths.

SEE ALSO

`trace_paths(2)`

timing_precharge_same_pchg_eval_clock_cont

Sets the minimum amount of time between the precharge and evaluate clock transitions when the same clock is used for both the precharge and evaluate phases.

TYPE

float

DEFAULT

0

DESCRIPTION

When the **timing_precharge_check_contention_between_same_pchg_eval_clock** variable is set to **true**, NanoTime checks for contention between the precharge and evaluate clock transitions of D1 type domino circuits when the same clock is used for the precharge and evaluate phases.

By default, when this type of checking is enabled, the amount of time between the precharge and evaluate clock transitions must be at least zero. To specify a different time constraint, set the **timing_precharge_same_pchg_eval_clock_contention_margin** variable to a nonzero value. The specified value is the minimum allowable time separation between the precharge and evaluate clock transitions, in nanoseconds.

Specify a positive value for a more conservative check that requires more time separation, to ensure that contention does not occur. Specify a negative value for a more relaxed check that allows some overlap between the precharge and evaluate phases.

SEE ALSO

```
trace_paths(2)
timing_precharge_check_contention_between_same_pchg_eval_clock(3)
```

timing_precharge_same_phase_from_non_prec

Selects same-phase or no-same-phase checking for an input signal at a precharge cell coming from a non-precharge driver.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable, when set to **true** (the default), causes an input signal at a precharge cell coming from a non-precharge driver to be considered a same-phase timing check. When the variable is set to **false**, this condition is considered a no-same-phase timing check.

With same-phase checking, for a setup check, data launched at the upstream latch by the opening edge of the launch clock must arrive at the downstream latch before the closing edge of the capture clock in the *same* clock phase. For a hold check, the same data launched at the upstream latch must arrive at the downstream latch no earlier than the closing edge of the capture clock in the *previous* clock phase.

With no-same-phase checking, for a setup check, data launched at the upstream latch by the opening edge of the launch clock must arrive at the downstream latch before the closing edge of the capture clock in the *next* clock phase. For a hold check, the same data launched at the upstream latch must arrive at the downstream latch no earlier than the closing edge of the capture clock in the *same* clock phase.

SEE ALSO

```
trace_paths(2)
```

timing_precharge_setup_fall_margin

Specifies the amount of timing margin for falling-edge setup checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge setup checks at the constrained (data) pin of precharge structures. The specified margin value applies to a falling-edge signal arriving at the constrained pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After executing the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_setup_fall_margin(3)
timing_flip_flop_setup_fall_margin(3)
timing_latch_setup_fall_margin(3)
timing_precharge_in_setup_fall_margin(3)
timing_ram_setup_fall_margin(3)
```

timing_precharge_setup_rise_margin

Specifies the amount of timing margin for rising-edge setup checks at precharge structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge setup checks at the constrained (data) pin of precharge structures. The specified margin value applies to a rising-edge signal arriving at the constrained pin of a precharge structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After executing the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After running the **trace_paths** command, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
check_design(2)
mark_precharge(2)
set_timing_check_attributes(2)
report_constraint(2)
trace_paths(2)
timing_clock_gate_setup_rise_margin(3)
timing_flip_flop_setup_rise_margin(3)
timing_latch_setup_rise_margin(3)
timing_precharge_in_setup_rise_margin(3)
timing_ram_setup_rise_margin(3)
```

timing_propagate_interclock_uncertainty

Enables or disables the propagation of interclock uncertainty through transparent latches.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the interclock uncertainty is calculated from the clock at the most recently traversed transparent latch to the clock at the capture latch, even when the origination of the path has a different clock.

If you set the **timing_propagate_interclock_uncertainty** variable to **true**, clock uncertainty information is propagated through each latch operating in transparent mode as though it were a combinational element. This allows a path with multiple latches to use the originating clock of the path as the launch clock, which is then compared to the capture clock to determine uncertainty.

For example, consider a pipeline containing latches A, B, and C clocked by clocks 1, 2, and 3, respectively. If latch B is in transparent mode, data passes through it as though it were a combinational element. Regardless of whether interclock uncertainty has been applied between clocks 1 and 3, the default behavior is to apply the uncertainty between clocks 2 and 3 when calculating slack at latch C.

However, you might want to apply the uncertainty between the clock at the path startpoint (clock 1, latch A) and the clock at the path endpoint (clock 3, latch C), if defined. If you set the **timing_propagate_interclock_uncertainty** variable to **true**, the interclock uncertainty is applied as though the path from latch A to latch C through the transparent latch B were a single, extended path. In other words, the uncertainty is propagated through the transparent latch.

If the path being analyzed is launched from an input port whose input delay is not specified relative to a

clock, the uncertainty calculation reverts to the default behavior. NanoTime uses the last traversed transparent latch for the launch clock and the path end for the capture clock.

SEE ALSO

report_paths(2)
set_clock_uncertainty(2)

timing_ram_hold_fall_margin

Specifies the amount of timing margin for falling-edge hold checks at RAM structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge hold checks at RAM structures. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any hold check at a RAM structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_ram_hold_rise_margin(3)
timing_ram_setup_fall_margin(3)
timing_ram_setup_rise_margin(3)
```

timing_ram_hold_rise_margin

Specifies the amount of timing margin for rising-edge hold checks at RAM structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge hold checks at RAM structures. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any hold check at a RAM structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_ram_hold_fall_margin(3)
timing_ram_setup_fall_margin(3)
timing_ram_setup_rise_margin(3)
```

timing_ram_setup_fall_margin

Specifies the amount of timing margin for falling-edge setup checks at RAM structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all falling-edge setup checks at RAM structures. The specified margin value applies to the falling-edge signal arriving at the constrained (data) pin of any setup check at a RAM structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_ram_hold_rise_margin(3)
timing_ram_hold_fall_margin(3)
timing_ram_setup_rise_margin(3)
```

timing_ram_setup_rise_margin

Specifies the amount of timing margin for rising-edge setup checks at RAM structures.

TYPE

float

DEFAULT

0.0

DESCRIPTION

This variable sets the amount of timing margin required for all rising-edge setup checks at RAM structures. The specified margin value applies to the rising-edge signal arriving at the constrained (data) pin of any setup check at a RAM structure.

By default, the margin is set to zero. Specify a positive value for a more conservative and restrictive check, or a negative value for a more relaxed and permissive check.

The variable must be set before you use the **check_design** command. After running the **check_design** command, you can adjust timing check margins at individual locations in the design by using the **set_timing_check_attributes** command.

After path tracing is complete, you can see the results of constraint evaluation by using the **report_constraint** command.

SEE ALSO

```
create_timing_check(2)
report_constraint(2)
set_timing_check_attributes(2)
timing_ram_hold_rise_margin(3)
timing_ram_hold_fall_margin(3)
timing_ram_setup_fall_margin(3)
```

timing_register_file_add_cross_delay_constrai

Enables the creation of delay arcs and setup/hold timing checks to the latch net on the opposite side of a register file.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to true, enables the creation of delay arcs from the driving gate of a register file to the latch net on the opposite side of the same register file. It will also allow setup/hold timing checks to be reported which check such a data path across the register file against the clock pin on either side of the register file. The default of the vairable is false, in which case only delay arcs and timing checks on the same side of a register file would be considered.

SEE ALSO

```
mark_register_file(2)
create_timing_check(2)
```

timing_same_phase_gated_clock

Sets same-phase or no-same-phase checking for gated-clock timing checks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, when a reference clock is propagated to a clock input pin of a gated clock, and the same reference clock triggers the enable signal of the gated clock, NanoTime performs a gated-clock timing check in the next clock cycle (no-same-phase checking). To perform the timing check in the same clock cycle (same-phase checking), set this variable to **true**.

SEE ALSO

```
remove_gated_clock_check(2)
create_gated_clock_timing_check(2)
set_gated_clock_timing_check_attributes(2)
```

timing_save_clock_paths

Determines whether NanoTime saves clock network timing paths into the path database during path tracing.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, during path tracing NanoTime saves clock network timing paths into the path database. Then you can use the **report_paths -clock_only** command to get detailed reports on the timing of clock paths from the clock input ports to the clock pins of sequential elements.

If you set the **timing_save_clock_paths** variable to **false**, clock network timing paths are not saved during path tracing, reducing the total memory usage.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

timing_save_pin_arrival_and_transition

Determines whether NanoTime saves arrival time and transition time information for every timing point in the design during path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, during path tracing, NanoTime saves worst-case data arrival information into the path database for path endpoints only.

When this variable is set to **true**, NanoTime saves the worst-case arrival times (max rise, max fall, min rise, and min fall) and slopes (longest rise, longest fall, shortest rise, shortest fall) for all timing points in the design.

You can get the detailed arrival and transition information from the path database by using the **get_attribute -class pin** command.

This feature can be used to find out whether a specific point in the design has been traced, or to find out the worst arrival times or slopes at intermediate points in a path.

SEE ALSO

```
get_attribute(2)
report_paths(2)
trace_paths(2)
```

timing_save_violating_internal_model_paths

Determines whether NanoTime saves timing paths at pins with timing violations into the path database during model path tracing from clocks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, during model path tracing from clocks NanoTime does not save timing paths at pins with timing violations. If they are saved by enabling this variable, you can use the **report_paths** command to get detailed reports on the timing of paths to pins where the violation occurred. Such paths are given the attribute **internal_modeling_violation**.

If you set the **timing_save_violating_internal_model_paths** variable to **true**, the additional timing paths saved increases the total memory usage.

For each timing violation at a pin, NanoTime saves one timing path which might get superceded by a more critical path as the tracing continues. The total number of paths to that pin depends on the number of clock start points from which the pin can be reached.

Timing violations from non-clock inputs are not covered by this variable.

SEE ALSO

```
report_paths(2)
extract_model(2)
```

timing_save_wire_delay

Determines whether NanoTime saves arrival times at both the beginning and end of each wire connection during path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime does not save detailed arrival times of RC networks during path tracing. To debug the timing of paths with detailed parasitics, set the **timing_save_wire_delay** variable to **true**. In that case, **report_paths** shows the arrival times at both the beginning and end of each RC tree. Note that this might cause a significant increase in memory usage and possibly runtime.

When the **timing_save_wire_delay** variable is set to **true**, the **report_arrivals** command also reports the arrival times at the beginning of each RC tree.

SEE ALSO

```
report_paths(2)
report_arrivals(2)
trace_paths(2)
```

timing_turnoff_delay_max_fall

Specifies the delay resulting from falling turnoff transitions at all three-state nodes during maximum-delay path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

By default, NanoTime calculates a three-state turnoff delay as the amount of time it takes for the driving transistor to go into the high-impedance state. The tool considers the transistor to be off when the gate-to-source voltage falls to the threshold voltage of the transistor.

To override this calculation globally and specify a delay value for all falling turnoff transitions during maximum-delay path tracing, set this variable to the desired time value. A falling turnoff transition is a change from logic 1 to the high-impedance logic-low state at the turnoff node.

Setting this variable to zero (the default setting) disables the override feature and allows NanoTime to calculate the delay.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

```
timing_turnoff_delay_max_rise(3)
timing_turnoff_delay_min_fall(3)
timing_turnoff_delay_min_rise(3)
```

timing_turnoff_delay_max_rise

Specifies the delay resulting from rising turnoff transitions at all three-state nodes during maximum-delay path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

By default, NanoTime calculates a three-state turnoff delay as the amount of time it takes for the driving transistor to go into the high-impedance state. The tool considers the transistor to be off when the gate-to-source voltage falls to the threshold voltage of the transistor.

To override this calculation globally and specify a delay value for all rising turnoff transitions during maximum-delay path tracing, set this variable to the desired time value. A rising turnoff transition is a change from logic 0 to the high-impedance logic-high state at the turnoff node.

Setting this variable to zero (the default setting) disables the override feature and allows NanoTime to calculate the delay.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

```
timing_turnoff_delay_max_fall(3)
timing_turnoff_delay_min_fall(3)
timing_turnoff_delay_min_rise(3)
```

timing_turnoff_delay_min_fall

Specifies the delay resulting from falling turnoff transitions at all three-state nodes during minimum-delay path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

By default, NanoTime calculates a three-state turnoff delay as the amount of time it takes for the driving transistor to go into the high-impedance state. The tool considers the transistor to be off when the gate-to-source voltage falls to the threshold voltage of the transistor.

To override this calculation globally and specify a delay value for all falling turnoff transitions during minimum-delay path tracing, set this variable to the desired time value. A falling turnoff transition is a change from logic 1 to the high-impedance logic-low state at the turnoff node.

Setting this variable to zero (the default setting) disables the override feature and allows NanoTime to calculate the delay.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

```
timing_turnoff_delay_max_fall(3)
timing_turnoff_delay_max_rise(3)
timing_turnoff_delay_min_rise(3)
```

timing_turnoff_delay_min_rise

Specifies the delay resulting from rising turnoff transitions at all three-state nodes during minimum-delay path tracing.

TYPE

float

DEFAULT

0.0

DESCRIPTION

By default, NanoTime calculates a three-state turnoff delay as the amount of time it takes for the driving transistor to go into the high-impedance state. The tool considers the transistor to be off when the gate-to-source voltage falls to the threshold voltage of the transistor.

To override this calculation globally and specify a delay value for all rising turnoff transitions during minimum-delay path tracing, set this variable to the desired time value. A rising turnoff transition is a change from logic 0 to the high-impedance logic-high state at the turnoff node.

Setting this variable to zero (the default setting) disables the override feature and allows NanoTime to calculate the delay.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

```
timing_turnoff_delay_max_fall(3)
timing_turnoff_delay_max_rise(3)
timing_turnoff_delay_min_fall(3)
```

timing_turnoff_max_propagation

Determines whether to propagate forward the turnoff delay at three-state nodes during maximum-delay path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, during path tracing, NanoTime computes the turnoff delay at a three-state output of a cell but does not propagate the delay forward. If the three-state output is a checked point or an output port of the design, NanoTime saves the path into the path database, with the three-state node as the path endpoint (subject to any limitation on the number of paths that can be saved). A checked point is a location where a timing check is evaluated, such as a latch input.

If the **timing_turnoff_max_propagation** variable is set to **true**, NanoTime considers three-state transitions to be valid at intermediate points of a maximum-delay path, so it propagates these transitions forward to the next point in the path.

This feature can be useful for a "contention recovery" type signal. When a three-state controlling signal is turning off, another three-state driver could be turning on. If there is some overlap between the second driver turning on and the first controlling signal turning off, then the actual signal propagates forward at the moment the first controlling signal "releases" the output signal. This situation calls for the turnoff signal to be propagated forward.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_turnoff_min_propagation(3)
```

timing_turnoff_min_propagation

Determines whether to propagate forward the turnoff delay at three-state nodes during minimum-delay path tracing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, during path tracing, NanoTime computes the turnoff delay at a three-state output of a cell but does not propagate the delay forward. If the three-state output is a checked point or an output port of the design, NanoTime saves the path into the path database, with the three-state node as the path endpoint (subject to any limitation on the number of paths that can be saved). A checked point is a location where a timing check is evaluated, such as a latch input.

If the **timing_turnoff_min_propagation** variable is set to **true**, NanoTime considers three-state transitions to be valid at intermediate points of a minimum-delay path, so it propagates these transitions forward to the next point in the path.

This feature can be useful for a "contention recovery" type signal. When a three-state controlling signal is turning off, another three-state driver could be turning on. If there is some overlap between the second driver turning on and the first controlling signal turning off, then the actual signal propagates forward at the moment the first controlling signal "releases" the output signal. This situation calls for the turnoff signal to be propagated forward.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_turnoff_max_propagation(3)
```

timing_turnoff_transition_max_fall

Specifies the transition time resulting from falling turnoff transitions at all three-state nodes, for maximum-delay path tracing.

TYPE

float

DEFAULT

0.05

DESCRIPTION

For maximum-delay path tracing, NanoTime propagates the turnoff delay at the output of a three-state cell forward through a path if the **timing_turnoff_max_propagation** variable is set to true. For a falling turnoff transition (from logic 1 to the high-impedance logic-low state at the turnoff node), the turnoff transition time that is propagated is set by the **timing_turnoff_transition_max_fall** variable.

The **timing_turnoff_transition_max_fall** variable is set to 0.05 by default. The minimum value is 3ps (0.003); if you set the variable to any value less than 0.003, NanoTime uses a value of 0.003.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_turnoff_max_propagation(3)
timing_turnoff_transition_max_rise(3)
timing_turnoff_transition_min_fall(3)
timing_turnoff_transition_min_rise(3)
```

timing_turnoff_transition_max_rise

Specifies the transition time resulting from rising turnoff transitions at all three-state nodes, for maximum-delay path tracing.

TYPE

float

DEFAULT

0.05

DESCRIPTION

For maximum-delay path tracing, NanoTime propagates the turnoff delay at the output of a three-state cell forward through a path if the **timing_turnoff_max_propagation** variable is set to true. For a rising turnoff transition (from logic 0 to the high-impedance logic-high state at the turnoff node), the turnoff transition time that is propagated is determined by the **timing_turnoff_transition_max_rise** setting.

The **timing_turnoff_transition_max_rise** variable is set to 0.05 by default. The minimum value is 3ps (0.003); if you set the variable to any value less than 0.003, NanoTime uses a value of 0.003.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_turnoff_max_propagation(3)
timing_turnoff_transition_max_fall(3)
timing_turnoff_transition_min_fall(3)
timing_turnoff_transition_min_rise(3)
```

timing_turnoff_transition_min_fall

Specifies the transition time resulting from falling turnoff transitions at all three-state nodes, for minimum-delay path tracing.

TYPE

float

DEFAULT

0.05

DESCRIPTION

For minimum-delay path tracing, NanoTime propagates the turnoff delay at the output of a three-state cell forward through a path if the **timing_turnoff_min_propagation** variable is set to true. For a falling turnoff transition (from logic 1 to the high-impedance logic-low state at the turnoff node), the turnoff transition time that is propagated is determined by the **timing_turnoff_transition_min_fall** setting.

The **timing_turnoff_transition_min_fall** variable is set to 0.05 by default. The minimum value is 3ps (0.003); if you set the variable to any value less than 0.003, NanoTime uses a value of 0.003.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_turnoff_min_propagation(3)
timing_turnoff_transition_max_fall(3)
timing_turnoff_transition_max_rise(3)
timing_turnoff_transition_min_rise(3)
```

timing_turnoff_transition_min_rise

Specifies the transition time resulting from rising turnoff transitions at all three-state nodes, for minimum-delay path tracing.

TYPE

float

DEFAULT

0.05

DESCRIPTION

For minimum-delay path tracing, NanoTime propagates the turnoff delay at the output of a three-state cell forward through a path if the **timing_turnoff_min_propagation** variable is set to true. For a rising turnoff transition (from logic 1 to the high-impedance logic-high state at the turnoff node), the turnoff transition time that is propagated is determined by the **timing_turnoff_transition_min_rise** setting.

The **timing_turnoff_transition_min_rise** variable is set to 0.05 by default. The minimum value is 3ps (0.003); if you set the variable to any value less than 0.003, NanoTime uses a value of 0.003.

SEE ALSO

```
report_paths(2)
trace_paths(2)
timing_turnoff_min_propagation(3)
timing_turnoff_transition_max_fall(3)
timing_turnoff_transition_max_rise(3)
timing_turnoff_transition_min_fall(3)
```

timing_turnon_output_delay_arcs

Specifies whether to make a distinction between timing arcs to three-state output arcs and to ordinary output timing arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime makes no distinction between timing arcs that end at ordinary outputs and those that end at three-state (high-impedance state) outputs. You can optionally have NanoTime consider these types of timing arcs as different. In that case, NanoTime reports enable and disable timing action in a different manner for each type of timing arc.

To have NanoTime consider these types of timing arcs as different, set the **timing_turnon_output_delay_arcs** variable to **true**. This setting affects the creation of timing arcs by the **extract_model** command and the path tracing done by the **trace_paths** command.

SEE ALSO

extract_model(2)
trace_paths(2)

timing_unclocked_latch_transparency

Forces all unclocked latches to be transparent.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Typically, a synchronizing clock waveform is available at each latch clock pin, which NanoTime uses for evaluating cycle-based timing checks. However, when the clock pin is driven by a data signal and not by a clock signal, this type of checking cannot be done. By default, when the NanoTime path tracer reaches the data pin of an unclocked latch, it stops tracing the path and generates a warning message.

If you want to find paths under the assumption that all unclocked latches are in the transparent state, set the **timing_unclocked_latch_transparency** variable to **true**. In that case, each unclocked latch behaves as if its clock pin is tied to a constant logic level that forces the latch to be transparent. At each of these latches, the latch timing checks are ignored and path tracing continues through the latch.

SEE ALSO

```
set_no_timing_check_path(2)
set_case_analysis(2)
trace_paths(2)
```

topo_allow_latch_on_clock_nets

Enables NanoTime to find latches on clock nets and perform latch timing checks on such latches.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime does not recognize latches on clock nets. It assumes that a clock does not propagate through a latch structure. In the case of clock divider, user can force the clock through but the latch topology will be dropped and no timing check is performed.

However, in some special cases, this assumption may not hold true. User can set \fbtopo_allow_latch_on_clock_nets to true to make NanoTime search for latch structures on clock nets and perform necessary timing checks. If the latch input net is a clock net too, user need to turn \fbtopo_find_clock_driven_data_inputs as well.

SEE ALSO

```
match_topology(2)
check_topology(2)
topo_find_clock_driven_data_inputs(3)
```

topo_auto_find_bit_column_mux

Enables automatic recognition of column muxes in memory topologies.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

NanoTime for memories can automatically detect column muxes connected to the bitlines. NanoTime recognizes single and multi level muxes only if the mux is composed entirely of PMOS or NMOS transistors. It does not recognize mixed topologies. The **topo_auto_find_bit_column_mux** variable enables the automatic recognition of such mux topologies.

If the **topo_auto_find_bit_column_mux** variable is set to true (the default), then NanoTime for memories checks for the presence of a column mux during topology recognition. If found, the structure is recognized as a mux. You can override such a mux by using the **erase_mux** command.

The **topo_auto_find_bit_column_mux** variable controls topology recognition. For this reason, if set after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

`mark_mux(2)`

```
match_topology(2)
erase_mux(2)
topo_ram_drive_res_ratio(3)
```

topo_auto_find_latch_clock

Allows the **mark_latch** command to be used without specifying a clock with the **-clock** option.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, if a **mark_latch** command does not include the **-clock** option, NanoTime assumes it is an oversight. It reports that the clock pin has not been specified and does not take any action.

To have NanoTime process **mark_latch** commands without the **-clock** option, set the **topo_auto_find_latch_clock** variable to **true**.

By default, NanoTime only searches non latch feedback devices as clock pins. To direct NanoTime to mark latch feedback devices as clock pins, set the **topo_auto_find_latch_feedback_clock** variable to **true**.

Note that NanoTime cannot determine the clock pins unless the connected nets have been classified as clock nets. For this reason, it is recommended that you specify the clock pins in **mark_latch** commands whenever possible.

The variable must be set before the **match_topology** command (or before the **check_topology** command if there is no **match_topology** command).

If there is more than one **match_topology** command and the variable is set after the first **match_topology** command, the variable has an effect only if NanoTime repropagates the clock network and rerecognizes topologies. This occurs only when two conditions are met:

1. The **topo_match_topology_reset_clock_and_topology** variable is set to **true**.

2. A subsequent **match_topology** command includes the **-force** option.

SEE ALSO

```
mark_latch(2)
match_topology(2)
topo_auto_find_latch_feedback_clock(3)
```

topo_auto_find_latch_clock_thru_port

Allows the **mark_latch** command to be used without specifying a clock with the **-clock** option.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When a **mark_latch** command does not include the **-clock** option and the **topo_auto_find_latch_clock** variable is set to **true**, NanoTime finds the clock pins for the latch automatically. By default, the clock pin search stops at ports. Set the **topo_auto_find_latch_clock_thru_port** variable to **true** to allow the clock pin search to continue through ports.

This variable affects clock propagation and latch and precharge recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
mark_latch(2)
match_topology(2)
topo_auto_find_latch_clock(3)
```

topo_auto_find_latch_feedback_clock

Allows latch feedback devices to be marked as latch clock pins.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, when automatically searching for latch clock pins, NanoTime only searches devices that are not latch feedback devices. To allow feedback devices to be marked as clock pins, set the **topo_auto_find_latch_feedback_clock** variable to **true**.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
mark_latch(2)
match_topology(2)
topo_auto_find_latch_clock(3)
```

topo_auto_recognize_clock_dependent_pattern

Overrides the default script used to implement automatic recognition and marking of clock-dependent structures in the design.

TYPE

string

DEFAULT

""

DESCRIPTION

The **read_pattern** and **foreach_match** commands can find and mark structures in the design based on pattern matching. You can automate the process by placing pattern-matching scripts into a directory. A script file includes a **read_pattern** command that reads in a SPICE specification of the pattern and a **foreach_match** command to carry out the desired operation on all occurrences of the pattern.

By default, NanoTime executes the following pattern-matching scripts during the **match_topology** command:

$synopsys_root/auxx/nt/patterns/patterns.tcl

$synopsys_root/auxx/nt/patterns/clock_dependent_patterns.tcl

where $synopsys_root is the NanoTime installation directory. The patterns.tcl script applies to clock-independent structures such as inverters, multiplexers, and transfer gates. The clock_dependent_patterns.tcl script applies to clocked structures such as latches and precharge circuits.

Under unusual circumstances, you might want to override the default scripts. To do so, set the following variables to a different directory and script file name:

For clock-independent structures, set the **topo_auto_recognize_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_clock_dependent_pattern_command_file** variable.

To add your own pattern-matching script to be executed during the **match_topology** command, set the following variables to point to the new script files. Both the default scripts and your new scripts are executed.

For clock-independent structures, set the **topo_auto_recognize_user_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_user_clock_dependent_pattern_command_file** variable.

For example:

```
nt_shell> set topo_auto_recognize_user_pattern_command_file \
          /user/mydir/patterns2.tcl
```

SEE ALSO

foreach_match(2)
match_topology(2)
read_pattern(2)
topo_auto_recognize_user_clock_dependent_pattern_command_file(3)
topo_auto_recognize_pattern_command_file(3)
topo_auto_recognize_user_pattern_command_file(3)

topo_auto_recognize_pattern_command_file

Overrides the default script used to implement automatic recognition and marking of clock-independent structures in the design.

TYPE

string

DEFAULT

""

DESCRIPTION

The **read_pattern** and **foreach_match** commands can find and mark structures in the design based on pattern matching. You can automate the process by placing pattern-matching scripts into a directory. A script file includes a **read_pattern** command that reads in a SPICE specification of the pattern and a **foreach_match** command to carry out the desired operation on all occurrences of the pattern.

By default, NanoTime executes the following pattern-matching scripts during the **match_topology** command:

$synopsys_root/auxx/nt/patterns/patterns.tcl

$synopsys_root/auxx/nt/patterns/clock_dependent_patterns.tcl

where $synopsys_root is the NanoTime installation directory. The patterns.tcl script applies to clock-independent structures such as inverters, multiplexers, and transfer gates. The clock_dependent_patterns.tcl script applies to clocked structures such as latches and precharge circuits.

Under unusual circumstances, you might want to override the default scripts. To do so, set the following variables to a different directory and script file name:

For clock-independent structures, set the **topo_auto_recognize_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_clock_dependent_pattern_command_file** variable.

To add your own pattern-matching script to be executed during the **match_topology** command, set the following variables to point to the new script files. Both the default scripts and your new scripts are executed.

For clock-independent structures, set the **topo_auto_recognize_user_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_user_clock_dependent_pattern_command_file** variable.

For example:

```
nt_shell> set topo_auto_recognize_user_pattern_command_file \
          /user/mydir/patterns2.tcl
```

SEE ALSO

foreach_match(2)
match_topology(2)
read_pattern(2)
topo_auto_recognize_user_clock_dependent_pattern_command_file(3)
topo_auto_recognize_clock_dependent_pattern_command_file(3)
topo_auto_recognize_user_pattern_command_file(3)

topo_auto_recognize_user_clock_dependent_pa

Specifies a script file used to implement automatic recognition and marking of clock-dependent structures in the design.

TYPE

string

DEFAULT

""

DESCRIPTION

The **read_pattern** and **foreach_match** commands can find and mark structures in the design based on pattern matching. You can automate the process by placing pattern-matching scripts into a directory. A script file includes a **read_pattern** command that reads in a SPICE specification of the pattern and a **foreach_match** command to carry out the desired operation on all occurrences of the pattern.

By default, NanoTime executes the following pattern-matching scripts during the **match_topology** command:

$synopsys_root/auxx/nt/patterns/patterns.tcl

$synopsys_root/auxx/nt/patterns/clock_dependent_patterns.tcl

where $synopsys_root is the NanoTime installation directory. The patterns.tcl script applies to clock-independent structures such as inverters, multiplexers, and transfer gates. The clock_dependent_patterns.tcl script applies to clocked structures such as latches and precharge circuits.

Under unusual circumstances, you might want to override the default scripts. To do so, set the following variables to a different directory and script file name:

For clock-independent structures, set the **topo_auto_recognize_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_clock_dependent_pattern_command_file** variable.

To add your own pattern-matching script to be executed during the **match_topology** command, set the following variables to point to the new script files. Both the default scripts and your new scripts are executed.

For clock-independent structures, set the **topo_auto_recognize_user_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_user_clock_dependent_pattern_command_file** variable.

For example:

```
nt_shell> set topo_auto_recognize_user_pattern_command_file \
          /user/mydir/patterns2.tcl
```

SEE ALSO

foreach_match(2)
match_topology(2)
read_pattern(2)
topo_auto_recognize_clock_dependent_pattern_command_file(3)
topo_auto_recognize_pattern_command_file(3)
topo_auto_recognize_user_pattern_command_file(3)

topo_auto_recognize_user_pattern_command_f

Specifies a script file used to implement automatic recognition and marking of clock-independent structures in the design.

TYPE

string

DEFAULT

""

DESCRIPTION

The **read_pattern** and **foreach_match** commands can find and mark structures in the design based on pattern matching. You can automate the process by placing pattern-matching scripts into a directory. A script file includes a **read_pattern** command that reads in a SPICE specification of the pattern and a **foreach_match** command to carry out the desired operation on all occurrences of the pattern.

By default, NanoTime executes the following pattern-matching scripts during the **match_topology** command:

$synopsys_root/auxx/nt/patterns/patterns.tcl

$synopsys_root/auxx/nt/patterns/clock_dependent_patterns.tcl

where $synopsys_root is the NanoTime installation directory. The patterns.tcl script applies to clock-independent structures such as inverters, multiplexers, and transfer gates. The clock_dependent_patterns.tcl script applies to clocked structures such as latches and precharge circuits.

Under unusual circumstances, you might want to override the default scripts. To do so, set the following variables to a different directory and script file name:

For clock-independent structures, set the **topo_auto_recognize_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_clock_dependent_pattern_command_file** variable.

To add your own pattern-matching script to be executed during the **match_topology** command, set the following variables to point to the new script files. Both the default scripts and your new scripts are executed.

For clock-independent structures, set the **topo_auto_recognize_user_pattern_command_file** variable.

For clock-dependent structures, set the **topo_auto_recognize_user_clock_dependent_pattern_command_file** variable.

For example:

```
nt_shell> set topo_auto_recognize_user_pattern_command_file \
          /user/mydir/patterns2.tcl
```

SEE ALSO

foreach_match(2)
match_topology(2)
read_pattern(2)
topo_auto_recognize_user_clock_dependent_pattern_command_file(3)
topo_auto_recognize_clock_dependent_pattern_command_file(3)
topo_auto_recognize_pattern_command_file(3)

topo_auto_search_class

Specifies the topology classes that are searched by default.

TYPE

string

DEFAULT

mux clock_gate turnoff xor cross_coupled nand nor latch precharge ram feedback weak_pullup differential_synchronizer

DESCRIPTION

This global variable controls which built-in topology classes are enabled for automatic topology searching. Automatic searching occurs when you use **match_topology** without the **-name** or **-structure_types** option.

This variable controls topology recognition. For this reason, if you set this variable after the **match_topology** command, it is recommended that you use the **match_topology -force** command to re-propagate the clock network and re-recognize topologies.

SEE ALSO

`check_topology(2)`
`match_topology(2)`

topo_big_ccb_transistors

Sets the maximum number of transistors in a channel-connected block (CCB) for the CCB to be considered during storage node recognition.

TYPE

integer

DEFAULT

100

DESCRIPTION

This variable specifies the largest number of transistors in a CCB for the CCB to be considered in storage node recognition. NanoTime automatically recognizes storage nodes by detecting CCB loops. If the number of transistors in a CCB is more than the value of the **topo_big_ccb_transistors** variable, NanoTime ignores the CCB during loop detection.

SEE ALSO

`report_storage_node(2)`

topo_ccb_max_number_of_inputs

Specifies the maximum number of inputs allowed for a channel-connected region.

TYPE

integer

DEFAULT

1000

DESCRIPTION

To facilitate correct timing graph construction and avoid excessive runtime, NanoTime limits the number of inputs to a channel-connected region to this value. Anything exceeding this limit is ignored during analysis. If valid channel-connected regions exist with a larger number of inputs, set the **topo_ccb_max_number_of_inputs** variable to a value greater than the largest number of inputs.

SEE ALSO

`topo_big_ccb_transistors(3)`

topo_check_clock_network

Specifies whether to strictly check the clock network at the **check_topology** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime can perform an optional check on the clock network at the **check_topology** or **match_topology** command. Two types of error conditions are detected:

```
(1) The clock network stops at invalid endpoints.
(2) Some nets which are required to be clocks are not part of a clock network.
```

By default, the check on the clock network is disabled. To enable the check, set the **topo_check_clock_network** variable to **true**.

SEE ALSO

```
check_topology(2)
set_requires_clock(2)
report_clock_network(2)
report_channel_connected_block(2)
report_storage_node(2)
```

topo_check_storage_node

Specifies whether to strictly check storage nodes during topology checking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the storage node check is off. To turn on the check, set the **topo_check_storage_node** variable to **true**. The **check_topology** command returns an error if the NanoTime tool finds storage nodes in error conditions.

Two types of error conditions are detected: 1) the storage node is not marked as a sequential topology, or 2) the clock does not propagate to the expected sequential device clock pins.

SEE ALSO

```
check_topology(2)
remove_storage_node(2)
report_channel_connected_block(2)
report_storage_node(2)
```

topo_check_transistor_direction

Specifies whether to strictly check transistor direction during topology checking.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime checks transistor direction during topology checking. The NanoTime tool returns an error at the **check_topology** command if it finds bidirectional transistors that are not user-specified. To disable this checking, set the **topo_check_transistor_direction** variable to **false**.

SEE ALSO

`check_topology(2)`

topo_clock_gate_allow_reconvergent_clocks

Specifies whether to recognize pulse generator and pulse shaper clock-gating topologies implemented as reconvergent clocks.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, when NanoTime identifies clock-gating structures during topology recognition, it assumes that a single clock signal and any number of enable signals enter the structure. However, some clock-gating structures, such as pulse generator and pulse shaper circuits, receive two clock signals with the enable signals.

To have NanoTime recognize these types of structures, set the **topo_clock_gate_allow_reconvergent_clocks** variable to true. In that case, NanoTime also recognizes clock-gating structures that receive two clock signals originating from the same clock point, possibly having different delays, and possibly with one signal inverted with respect to the other. If one signal is delayed (but not inverted) with respect to the other, NanoTime considers the structure a pulse shaper. If one signal is inverted with respect to the other, NanoTime considers the structure a pulse generator. After topology recognition, a verbose topology report (**report_topology -structure_type clock_gate -verbose**) reports a pulse generator attribute as "p" and a pulse shaper attribute as "s."

This variable controls topology recognition. For this reason, if set after the **match_topology** command, it is recommended that you use the **match_topology -force** command to re-propagate the clock network and re-recognize topologies.

SEE ALSO

```
mark_clock_gate(2)
match_topology(2)
report_topology(2)
```

topo_clock_gate_depth

Controls the depth of clock gate propagation.

TYPE

integer

DEFAULT

5

DESCRIPTION

The **topo_clock_gate_depth** variable controls the number of static logic gates after which the clock flag does not propagate.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
check_topology(2)
mark_clock_gate(2)
match_topology(2)
```

topo_clock_gate_strict_checking

Specifies whether strict checking for reconvergent clock gate recognition is enabled.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime identifies the possibility that an upstream logic gate can block a certain clock input to a downstream reconvergent clock gate. When this happens, the downstream reconvergent clock gate is recognized as an unresolved pulse-shaper or unresolved pulse-generator in order to guarantee pessimism. This feature can be disabled by setting the **topo_clock_gate_strict_checking** variable to **false**, in which case NanoTime performs a more aggressive clock gate recognition and marks any such downstream clock gate to be a resolved pulse-shaper or resolved pulse-generator.

This variable affects clock gate topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
match_topology(2)
mark_clock_gate(2)
topo_clock_gate_allow_reconvergent_clocks(3)
```

topo_clock_gate_timing_resolution

Specifies if arrival time data should be used to resolve which pin is the controlling pin in a clock shaper.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The NanoTime tool can exploit arrival time information to find the controlling pins of a clock gating topology with multiple input clocks, if the clock gate has a series-parallel topology.

For example, the latest arrival time to a series chain is the one that finally turns on the chain as a whole. Therefore, the pin with this latest time is picked as the controlling pin of this topology.

SEE ALSO

`topo_clock_gate_strict_checking(3)`

topo_clock_propagation_strict_logic_check

Specifies whether to strictly check logic for clock propagation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime propagates clock signals without checking the logic values set by case analysis or logic constraints. To consider logic values during clock propagation, set the **topo_clock_propagation_strict_logic_check** variable to **true**.

This variable affects clock propagation and latch and precharge recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

`check_topology(2)`
`match_topology(2)`

topo_copy_clock_property_from_differential_ne

Specifies whether certain clock properties must be copied on to a differential net from its paired net.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

The **topo_copy_clock_property_from_differential_net** variable specifies whether the following clock properties must be copied on to a differential net from its paired net: force_propagation, stop_propagation and no_pulse. By default, NanoTime copies these clock properties when identifying the clock network during **match_topology**.

SEE ALSO

`mark_clock_network(2)`
`match_topology(2)`

topo_create_lib_topology

Enables NanoTime to save a user defined latch, precharge, or register_file topology into a lib_topology object.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables NanoTime to save the commands **mark_latch**, **mark_precharge** or **mark_register_file**, and the topologies marked by the commands, into lib_topology objects.

By default, the variable is set to **false**. Set the variable to **true**, so that when the **-pattern** option is used for **mark_latch**, **mark_precharge** or **mark_register_file** command, the command and the marked topology will be saved as a lib_topology object. The lib_topology object is stored in the directory which is specified by the variable **topo_create_lib_topology_directory**.

SEE ALSO

```
mark_latch(2)
mark_precharge(2)
mark_register_file(2)
topo_create_lib_topology_directory(3)
```

topo_create_lib_topology_directory

Specifies the directory in which NanoTime stores the lib_topology objects created from user-defined latch, precharge and register_file topologies.

TYPE

string

DEFAULT

./_user_lib

DESCRIPTION

By default, the **topo_create_lib_topology_directory** variable is set to **./_user_lib**. You can set the variable to a different directory name.

SEE ALSO

```
mark_latch(2)
mark_precharge(2)
mark_register_file(2)
topo_create_lib_topology(3)
```

topo_feedback_inv_ratio

Sets the resistance ratio used to recognize the forward and feedback inverters of an inverter loop.

TYPE

float

DEFAULT

1.0

DESCRIPTION

An inverter loop is a pair of inverters connected together in a loop, output to input. By default, when NanoTime encounters a latch with an inverter loop, it recognizes the inverter with smaller resistance (stronger drive) as the forward inverter and assumes that the weaker one is a feedback inverter, for path tracing purposes. The input of the forward inverter is the latch-in node, and its output is the latch-out node.

For this comparison, the resistance of an inverter is the total of the resistance values of the NMOS and PMOS transistors in the inverter.

If the latch-in node drives multiple buffers, the inverter that drives the feedback inverter might not drive anything else, and might not need much drive strength. If the forward inverter is weaker than the feedback inverter, NanoTime does not correctly recognize the forward and feedback relationship.

If you set the **topo_feedback_inv_ratio** variable to a value smaller than the default of 1.0, NanoTime reverses its behavior of recognizing the forward and feedback inverters, when the two inverter resistance values are sufficiently similar.

The NanoTime tool checks the following relationship:

```
Rlarge * ratio >= Rsmall
```

where Rlarge is the resistance of the larger-resistance inverter, Rsmall is the resistance of the smaller-resistance inverter, and ratio is the value of the the **topo_feedback_inv_ratio** variable.

If this equation is true, the stronger (smaller-resistance) inverter is assumed to be the forward inverter, the same as the default case. If the equation is false, the weaker (larger-resistance) inverter is assumed to be the forward inverter instead. The equation is always true for the default ratio of 1.0.

If you set the ratio to 0.9, the stronger inverter is assumed to be the forward inverter when the resistance values of the inverters differ by at least 10 percent. However, if the resistance values are similar (within 10 percent), the weaker inverter is assumed to be the forward inverter.

When the two inverters have identical drive strengths, they are assumed to form a RAM cell, regardless of the value of the **topo_feedback_inv_ratio** variable.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
check_topology(2)
match_topology(2)
topo_feedback_p_res_ratio(3)
topo_feedback_n_res_ratio(3)
```

topo_feedback_n_res_ratio

Controls the relative resistance requirement for recognizing an NMOS feedback transistor connected between the input and output of an inverter.

TYPE

float

DEFAULT

1.0

DESCRIPTION

NanoTime recognizes NMOS and PMOS feedback transistors connected between the input and output of an inverter. The tool removes feedback transistors from path searches to prevent looping through the circuit.

To be recognized as a feedback transistor, an NMOS transistor must meet two requirements: connectivity and resistance.

Connectivity: The NMOS transistor's gate is connected to the output of the inverter, its drain (or source) is connected to the input of the inverter, and its source (or drain) is connected to ground.

Resistance: The resistance of the NMOS transistor is greater than the resistance of the NMOS transistor of the forward inverter.

If the circuit is designed with an NMOS feedback transistor that is stronger (having less resistance) than the NMOS transistor of the forward inverter, NanoTime does not recognize the feedback transistor, by default. To recognize this as a feedback transistor, modify the **topo_feedback_n_res_ratio** variable, which determines the relative resistance threshold for recognizing NMOS feedback transistors.

The NanoTime tool checks the following relationship:

```
Rtrans > Rforward * ratio
```

where Rtrans is the resistance of the NMOS transistor in question, Rforward is the resistance of the NMOS transistor in the forward inverter, and ratio is the value of the **topo_feedback_n_res_ratio** variable.

If this equation is true, the transistor is question is recognized as a feedback transistor, which is eliminated from path tracing and is considered an active device during delay calculation. If the equation is false, the transistor is not recognized as a feedback transistor and is treated as a parasitic element.

The default ratio is 1.0, causing a direct comparison between the two NMOS resistance values. A smaller ratio such as 0.8 makes it easier for a transistor to be recognized as feedback transistor; its resistance only needs to exceed the resistance of the inverter's transistor multiplied by the specified factor. A ratio of 0.0 causes all NMOS transistors meeting the connectivity requirements to be recognized as feedback transistors, regardless of their resistance values.

If the path is written with the **write_spice** command, a feedback transistor appears in the main SPICE netlist section. A transistor not recognized as a feedback transistor appears in the parasitics section.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

match_topology(2)
topo_feedback_p_res_ratio(3)
topo_feedback_inv_ratio(3)

topo_feedback_p_res_ratio

Controls the relative resistance requirement for recognizing a PMOS feedback transistor connected between the input and output of an inverter.

TYPE

float

DEFAULT

1.0

DESCRIPTION

NanoTime recognizes NMOS and PMOS feedback transistors connected between the input and output of an inverter. The tool removes feedback transistors from path searches to prevent looping through the circuit.

To be recognized as a feedback transistor, a PMOS transistor must meet two requirements: connectivity and resistance.

Connectivity: The PMOS transistor's gate is connected to the output of the inverter, its drain (or source) is connected to the input of the inverter, and its source (or drain) is connected to Vdd.

Resistance: The resistance of the PMOS transistor is greater than the resistance of the PMOS transistor of the forward inverter.

If the circuit is designed with a PMOS feedback transistor that is stronger (having less resistance) than the PMOS transistor of the forward inverter, NanoTime does not recognize the feedback transistor, by default. To recognize this as a feedback transistor, modify the **topo_feedback_p_res_ratio** variable, which determines the relative resistance threshold for recognizing PMOS feedback transistors.

The NanoTime tool checks the following relationship:

```
Rtrans > Rforward * ratio
```

where Rtrans is the resistance of the PMOS transistor in question, Rforward is the resistance of the PMOS transistor in the forward inverter, and ratio is the value of the **topo_feedback_p_res_ratio** variable.

If this equation is true, the transistor is question is recognized as a feedback transistor, which is eliminated from path tracing and is considered an active device during delay calculation. If the equation is false, the transistor is not recognized as a feedback transistor and is treated as a parasitic element.

The default ratio is 1.0, causing a direct comparison between the two PMOS resistance values. A smaller ratio such as 0.8 makes it easier for a transistor to be recognized as feedback transistor; its resistance only needs to exceed the resistance of the inverter's transistor multiplied by the specified factor. A ratio of 0.0 causes all PMOS transistors meeting the connectivity requirements to be recognized as feedback transistors, regardless of their resistance values.

If the path is written with the **write_spice** command, a feedback transistor appears in the main SPICE netlist section. A transistor not recognized as a feedback transistor appears in the parasitics section.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

match_topology(2)
topo_feedback_n_res_ratio(3)
topo_feedback_inv_ratio(3)

topo_find_clock_driven_data_inputs

Enables NanoTime to automatically identify data inputs of a sequential topology driven by a clock signal.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime assumes that a data input to a sequential structure is always driven by a data signal. Any clock driven input will be recognized as clock pins of the sequential topology. However, in some special cases, this may not be true. For example, the data input of a clock divider circuit is driven by a clock.

User can set \fbtopo_find_clock_driven_data_inputs to true to enable NanoTime find such clock driven data inputs of sequential topologies.

SEE ALSO

```
match_topology(2)
check_topology(2)
topo_sequential_structure_strict_input_matching(3)
```

topo_find_parallel_stack

Allows automatic recognition of parallel stacks.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, NanoTime automatically recognizes parallel stacks. To disable automatic recognition of parallel stacks, set the **topo_find_parallel_stack** variable to **false**.

Topology recognition happens at the **match_topology** command. Two or more series transistor stacks are recognized as parallel if all of the following conditions are met:

```
1) each stack has the same number of transistors
2) the stack transistors are the same type (all NMOS or all PMOS)
3) the stack connects to the same two nets on the boundary
4) besides the nets on the boundary, other nets in the stack are internal
5) walking from one boundary net to the other, the ith transistor of every
   stack has the same gate pin connections
```

After recognition, one stack of the group of parallel stacks is selected as representative and other parallel stacks are filtered from path tracing.

SEE ALSO

```
match_topology(2)
```

topo_flip_flop_strict_slave_check

Specifies whether to strictly check for the presence of the slave latch net in the master latch feedforward channel-connected block.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, the slave latch net is required to be in the feedforward channel-connected block of the master latch. To allow NanoTime to identify flip-flop topology structures when this is not true, set the **topo_flip_flop_strict_slave_check** variable to **false**.

The variable controls topology recognition. For this reason, if you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

`mark_flip_flop(2)`
`match_topology(2)`
`topo_auto_search_class(3)`

topo_latch_enable_logic_propagation

Enables the propagation of logic values through latches.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable specifies whether NanoTime propagates logic constants to latch nets. This propagation is done by adding an 'equal' logic constraint from the input net to the latch net. Propagation is performed only when the latch input is in a fixed logic state.

This variable applies to the following types of latches:

- Latches in which the input net and latch net are connected by a clocked unidirectional pass transistor

- Stacked latches that do not include a set/reset input in the input stack

The variable must be set before executing the **check_design** command.

SEE ALSO

`check_design(2)`

topo_latch_find_input_thru_clocked_fet

Enables searching latch inputs through the clocked devices on latch nets.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime collects the channel-connected block (CCB) inputs of the latch net as the latch inputs. If the **topo_latch_find_input_thru_clocked_fet** variable is set to **true**, NanoTime searches clocked devices on the latch net and marks the clocked net as latch input.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

`match_topology(2)`

topo_latch_find_tapped_feedback

Enables searching for tapped feedback devices.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If a latch feedback device also drives signals outside the latch through a non-feedback path, it is called tapped feedback. The non-feedback path is a valid timing path. NanoTime should allow path tracing to continue searching through the non-feedback paths while preserving the feedback property.

When the **topo_latch_find_tapped_feedback** variable is set to **true**, NanoTime automatically searches for tapped feedbacks within the latch topology during topology checking. Set the **topo_latch_find_tapped_feedback** variable to **false** to turn off the automatic search. Specify the tapped feedbacks with **-tapped_feedback** option in the **mark_latch** command.

SEE ALSO

`mark_latch(2)`
`check_topology(2)`

topo_latch_find_ts_feedback

Enables searching of tristate feedback latches.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, the NanoTime tool searches for tristate feedback latches during topology recognition. To disable this search, set the **topo_latch_find_ts_feedback** variable to **false**.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
check_topology(2)
match_topology(2)
topo_auto_search_class(3)
```

topo_latch_hold_to

Specifies the location in a latch circuit where NanoTime performs hold checking.

TYPE

string

DEFAULT

latch_net

DESCRIPTION

This variable specifies where NanoTime performs hold checking in latch circuits. It can be set to either **latch_net** (the default) or **input**. Setting the variable to **input** causes NanoTime to check for the arrival of data at the latch input rather than the latch node, resulting in a more conservative check.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

mark_latch(2)
match_topology(2)
topo_latch_setup_to(3)

topo_latch_setup_to

Specifies the location in a latch circuit where NanoTime performs setup checking.

TYPE

string

DEFAULT

latch_net

DESCRIPTION

This variable specifies where NanoTime performs setup checking in latch circuits. It can be set to **latch_net** (the default), **input**, or **output**. Setting the variable to **output** causes NanoTime to check for the arrival of data at the latch output rather than the latch node, resulting in a more conservative check. Setting the variable to **input** results in a less restrictive check.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

mark_latch(2)
match_topology(2)
topo_latch_hold_to(3)

topo_latch_strict_clock_check

Specifies whether to strictly check for the presence of a clock during automatic recognition and checking of latch structures.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, a clock signal is required at each latch. During execution of the **match_topology** and **check_topology** commands, NanoTime checks for the presence of a clock signal at each identified latch structure.

To allow NanoTime to identify latch structures without the presence of a clock, set the **topo_latch_strict_clock_check** variable to **false**. You might want to do this if you have not yet defined the clocks with the **create_clock** command. However, this variable setting might give incorrect results because every latch needs a clock.

The **topo_latch_strict_clock_check** variable affects automatically recognized latches only. It does not affect latches marked with the **mark_latch** command. If you set this variable after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
check_topology(2)
create_clock(2)
mark_latch(2)
match_topology(2)
```

topo_library_order

Enables the topology libraries used for topology matching and specifies the order in which NanoTime searches for topologies.

TYPE

string

DEFAULT

global common

DESCRIPTION

This variable enables topology libraries used for searching and marking topologies in the design. Only the libraries listed in the variable string are used in the current NanoTime session. The order of the libraries listed determines the order in which NanoTime searches for the topologies.

By default, the variable is set to the string **global common**, so that only the global and common libraries are used. If you change the variable setting, you must do so before you use the **match_topology** or **check_topology** commands.

For example, to enable the global, common, and user1 topology libraries, set the variable as follows:

```
nt_shell> set topo_library_order "global common user1"
```

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
create_topology_library(2)
list_topology_library(2)
read_topology_library(2)
report_topology_library(2)
```

topo_match_topology_reset_clock_and_topolog

Specifies whether to repropagate the clock network and rerecognize topology when the **match_topology** command is executed multiple times.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime propagates the clock network and recognizes topologies only once, at the first occurrence of either the **match_topology** or **check_topology** command. The **check_topology** command does not propagate clocks or recognize topologies unless there is no **match_topology** command.

If there is more than one **match_topology** command, other commands executed after the first **match_topology** command do not affect clock propagation and topology recognition.

To change this behavior, set the **topo_match_topology_reset_clock_and_topology** variable to **true**. Every subsequent **match_topology** command that includes the **-force** option causes NanoTime to repropagate the clock network and rerecognize topologies.

Commands that appear after the **set topo_match_topology_reset_clock_and_topology true** statement and before a **match_topology -force** command affect clock propagation and topology recognition. Commands that appear after the first **match_topology** command but before the variable is set to **true** do not.

SEE ALSO

match_topology(2)
check_topology(2)

topo_mux_drive_res_ratio

Controls the relative resistance requirement for recognizing a mux.

TYPE

float

DEFAULT

1.0

DESCRIPTION

NanoTime automatically recognizes mux (multiplexer) structures based on the presence of parallel pass gates or transmission gates that are connected together at the source or drain. NanoTime uses mux information to prevent the tracing of false paths.

In order to be recognized as a mux, the pass gates or transfer gates must meet the following requirements:

1. Connectivity: The pass gates or transfer gates must share a common drain or source.

2. Resistance: By default, the resistance of the pass gates or transfer gate must be the same.

If a mux circuit is designed with pass gates or transfer gates with slightly different strengths, NanoTime does not recognize the mux by default. To enable proper recognition, set the **topo_mux_drive_res_ratio** variable to allow a specified amount of difference in strengths. The variable sets a relative resistance threshold for recognizing a mux.

To be specific, NanoTime recognizes a mux if the resistance of the weakest pass gate (largest resistance) divided by the resistance of the strongest pass gate (smallest resistance) is less than the specified ratio. The ratio must be set to a value greater than or equal to 1.0.

For example, if you set the variable to 1.2, the structure's largest resistance can be up to 20 percent larger

than the smallest resistance and still be recognized as a mux. The default is 1.0, which requires an exact match for mux recognition.

An alternative to setting this variable is to mark the mux structures explicitly with the **mark_mux** command.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
mark_mux(2)
match_topology(2)
topo_ram_drive_res_ratio(3)
```

topo_mux_strict_enable_rules

Specifies whether to check mux structures for appropriate logic constraints on the enable control logic.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

NanoTime automatically places logic constraints on the selection lines of each mux structure, whether the mux is recognized automatically or marked by the user with the **mark_mux** command. NanoTime can do this for simpler mux structures. For more complex structures, you might need to manually specify the constraints.

If the **topo_mux_strict_enable_rules** variable is set to true (the default), NanoTime checks each mux for the presence of appropriate logic constraints. If it cannot find these constraints, it cancels recognition of the mux and generates a warning message.

To allow recognition of muxes without strict enforcement of logic constraint checking on the mux selection lines, set this variable to false.

The variable controls topology recognition. For this reason, if you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

mark_mux(2)
match_topology(2)

topo_ram_drive_res_ratio

Controls the relative resistance requirement for recognizing an inverter loop as a RAM.

TYPE

float

DEFAULT

1.0

DESCRIPTION

An inverter loop is a pair of inverters connected together in a loop, output to input. By default, when NanoTime encounters an inverter loop with equal-strength inverters, it recognizes the structure as a RAM. NanoTime terminates a path search when it reaches a RAM.

In order to be recognized as a RAM, the strengths of the two inverters must be identical. If a RAM is designed with inverters of slightly different strengths, NanoTime does not recognize the RAM by default. To enable proper recognition, set the **topo_ram_drive_res_ratio** variable to allow a specified amount of difference in strengths. The variable sets a relative resistance threshold for recognizing RAM structures.

To be specific, NanoTime recognizes a RAM if the resistance of the weaker inverter (larger resistance) divided by the resistance of the stronger inverter (smaller resistance) is less than the specified ratio. The ratio must be set to a value greater than or equal to 1.0.

For example, if you set the variable to 1.10, the larger resistance can be up to 10 percent larger than the smaller resistance and still be recognized as a RAM. The default is 1.0, which requires an exact match for RAM recognition.

An alternative to setting this variable is to mark the RAM structures explicitly with the **mark_ram** command.

This variable affects topology recognition. If you set it after the **match_topology** command, you should

use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

mark_mux(2)
match_topology(2)
topo_mux_drive_res_ratio(3)

topo_ram_find_all_pulldowns

Enables pulldown recognition in register files.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **topo_auto_search_class** variable includes the **pulldown** structure type and the **topo_ram_find_all_pulldowns** variable is set to **true** (the default), NanoTime automatically recognizes all enable pass gates of the RAM or register file as pulldown topologies without considering the drivers of the pass gates.

When the **topo_ram_find_all_pulldowns** variable is set to **false**, the tool turns off the pulldown recognition if the enable pass gates are driven by pullup and pulldown structures, such as an inverter.

SEE ALSO

```
topo_auto_search_class(3)
mark_ram(2)
mark_register_file(2)
match_topology(2)
```

topo_ram_search_thru_cell

Enables path searching through RAM cells.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **topo_ram_search_thru_cell** variable is set to **true** (the default), NanoTime performs path searching through a RAM or register file structure. The setup check occurs on the closing edge of the clock enable signal. When the variable is set to **false**, path searching stops at a RAM or register file structure. The setup check becomes nontransparent and occurs at the opening edge of the clock enable signal. The variable must be set before the **match_topology** command to have an effect.

SEE ALSO

```
mark_ram(2)
mark_register_file(2)
match_topology(2)
```

topo_regfile_search_all_clock_pins

Enables the tool to find clock pins by searching all clocked devices channel-connected to the register file nets.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the **topo_regfile_search_all_clock_pins** variable is **true**, NanoTime finds the clock pins by searching all clocked devices channel-connected to the register file nets. When the **topo_regfile_search_all_clock_pins** variable is **false** (the default), the tool starts the clock pin search from the register file net and stops the search when it finds the first clocked device.

SEE ALSO

`mark_register_file(2)`
`match_topology(2)`

topo_sequential_structure_install_extended_tir

When set to true, this variable allows NanoTime to install an extended set of timing checks for sequential topologies. This is usually applied when there are clock driven data inputs for sequential topologies.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

NanoTime assumes that a data input to a sequential structure is always driven by a data signal. By default, NanoTime will ignore any clock driven data inputs and will not have timing checks on such inputs.

When **topo_sequential_structure_install_extended_timing_checks** is set to true, NanoTime will perform "cross product" timing checks for any clock driven data inputs. For example, for a latch whose data pin is driven by clk1 and clock pin driven by clk2, NanoTime will first treat clk1 as the input and constrain it using clk2 as reference clock and then vice versa. User should expect one of these timing checks to fail.

SEE ALSO

```
topo_find_clock_driven_data_inputs(3)
topo_sequential_structure_strict_input_matching(3)
```

topo_sequential_structure_strict_input_matchi

Enables the strict input matching flow for sequential topologies. Under this strict flow, NanoTime will check if the data input of a sequential topology is driven by a data signal. If not, check_topology will fail and error will be reported.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime assumes that a data input to a sequential structure is always driven by a data signal. Any clock driven input will be recognized as clock pins of the sequential topology. However, in some special cases, this may not be true. For example, the data input of a clock divider circuit is driven by a clock.

When **topo_sequential_structure_strict_input_matching** is set to false (its default value), NanoTime issues warnings when it detects that a data input of a sequential structure is driven by a clock. NanoTime then proceeds with analysis. When **topo_sequential_structure_strict_input_matching** is set to true, NanoTime reports an error on any occurance of a clock driven data input and fails check_topology.

SEE ALSO

```
mark_latch(2)
mark_register_file(2)
```

```
mark_precharge(2)
check_topology(2)
topo_find_clock_driven_data_inputs(3)
```

topo_tgate_mark_all_pairs

Enables marking of all NMOS-PMOS pairs as transfer gates, regardless of the logic between the gates.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **topo_tgate_mark_all_pairs** variable is **true** (the default), NanoTime marks all NMOS-PMOS pairs as transfer gates without considering the logic relationship between the gates of the pair. When the **topo_tgate_mark_all_pairs** variable is **false**, an NMOS-PMOS pair is marked as a transfer gate only if there is an inverter between the gates of the pair.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
mark_tgate(2)
check_topology(2)
match_topology(2)
```

topo_waive_nondirectional_transistor

Specifies whether to waive transistor direction checking on nondirectional transistors during the **check_topology** operation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The NanoTime tool sets transistors to nondirectional if **set_case_analysis** or **set_logic_constraint** commands turn off the signal propagation flow through the transistor. By default, the tool considers such nondirectional transistors as direction unresolved when checking transistor direction during the **check_topology** operation.

To waive the direction checking, set the **topo_waive_nondirectional_transistor** variable to **true**.

SEE ALSO

```
check_topology(2)
report_transistor_direction(2)
set_transistor_direction(2)
```

topo_weak_pullup_n_length

Controls the length or width to length ratio requirement for recognizing NMOS weak pullup transistors.

TYPE

float

DEFAULT

0.0

DESCRIPTION

NanoTime recognizes NMOS and PMOS weak pullup transistors. An NMOS weak pullup transistor has its drain connected to Vdd, its gate tied to logic 1, and its source connected to the pullup node. The transistor is turned on, but it is weak enough to allow a stronger transistor to pull the node down to a low voltage. NanoTime ignores weak pullup transistors during logic propagation.

To determine whether an NMOS transistor is weak enough to be considered a weak pullup, NanoTime considers the following variables:

topo_weak_pullup_n_length
topo_weak_pullup_n_width

If both variables are set to zero (the default), automatic recognition of NMOS weak pullup transistors is disabled. (However, you can still identify weak pullup transistors with the **mark_weak_pullup** command.)

If both variables are set to nonzero values, NanoTime compares the width to length ratio of each NMOS transistor connected in a pullup configuration with the width to length ratio specified by the two variables. If the transistor is weaker (has a lower width to length ratio), it is considered a weak pullup. Otherwise, it is not considered a weak pullup and NanoTime considers the drive strength during logic propagation.

If only the length variable is set to a nonzero value, an NMOS transistor connected in a pullup configuration is considered a weak pullup if its length is more than the variable-specified value.

Similarly, if only the width variable is set to a nonzero value, an NMOS transistor connected in a pullup configuration is considered a weak pullup if its width is less than the variable-specified value.

You can override the weak or not-weak pullup determination for any transistor by using the **erase_weak_pullup** or **mark_weak_pullup** commands.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
erase_weak_pullup(2)
mark_weak_pullup(2)
match_topology(2)
topo_weak_pullup_n_width(3)
topo_weak_pullup_p_length(3)
topo_weak_pullup_p_width(3)
```

topo_weak_pullup_n_width

Controls the width or width to length ratio requirement for recognizing NMOS weak pullup transistors.

TYPE

float

DEFAULT

0.0

DESCRIPTION

NanoTime recognizes NMOS and PMOS weak pullup transistors. An NMOS weak pullup transistor has its drain connected to Vdd, its gate tied to logic 1, and its source connected to the pullup node. The transistor is turned on, but it is weak enough to allow a stronger transistor to pull the node down to a low voltage. NanoTime ignores weak pullup transistors during logic propagation.

To determine whether an NMOS transistor is weak enough to be considered a weak pullup, NanoTime considers the following variables:

topo_weak_pullup_n_length
topo_weak_pullup_n_width

If both variables are set to zero (the default), automatic recognition of NMOS weak pullup transistors is disabled. (However, you can still identify weak pullup transistors with the **mark_weak_pullup** command.)

If both variables are set to nonzero values, NanoTime compares the width to length ratio of each NMOS transistor connected in a pullup configuration with the width to length ratio specified by the two variables. If the transistor is weaker (has a lower width to length ratio), it is considered a weak pullup. Otherwise, it is not considered a weak pullup and NanoTime considers the drive strength during logic propagation.

If only the length variable is set to a nonzero value, an NMOS transistor connected in a pullup configuration is considered a weak pullup if its length is more than the variable-specified value.

Similarly, if only the width variable is set to a nonzero value, an NMOS transistor connected in a pullup configuration is considered a weak pullup if its width is less than the variable-specified value.

You can override the weak or not-weak pullup determination for any transistor by using the **erase_weak_pullup** or **mark_weak_pullup** commands.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
erase_weak_pullup(2)
mark_weak_pullup(2)
match_topology(2)
topo_weak_pullup_n_length(3)
topo_weak_pullup_p_length(3)
topo_weak_pullup_p_width(3)
```

topo_weak_pullup_p_length

Controls the length or width to length ratio requirement for recognizing PMOS weak pullup transistors.

TYPE

float

DEFAULT

0.0

DESCRIPTION

NanoTime recognizes NMOS and PMOS weak pullup transistors. A PMOS weak pullup transistor has its source connected to Vdd, its gate tied to logic 0, and its drain connected to the pullup node. The transistor is turned on, but it is weak enough to allow a stronger transistor to pull the node down to a low voltage. NanoTime ignores weak pullup transistors during logic propagation.

To determine whether a PMOS transistor is weak enough to be considered a weak pullup, NanoTime considers the following variables:

topo_weak_pullup_p_length
topo_weak_pullup_p_width

If both variables are set to zero (the default), automatic recognition of PMOS weak pullup transistors is disabled. (However, you can still identify weak pullup transistors with the **mark_weak_pullup** command.)

If both variables are set to nonzero values, NanoTime compares the width to length ratio of each PMOS transistor connected in a pullup configuration with the width to length ratio specified by the two variables. If the transistor is weaker (has a lower width to length ratio), it is considered a weak pullup. Otherwise, it is not considered a weak pullup and NanoTime considers the drive strength during logic propagation.

If only the length variable is set to a nonzero value, a PMOS transistor connected in a pullup configuration is considered a weak pullup if its length is more than the variable-specified value.

Similarly, if only the width variable is set to a nonzero value, a PMOS transistor connected in a pullup configuration is considered a weak pullup if its width is less than the variable-specified value.

You can override the weak or not-weak pullup determination for any transistor by using the **erase_weak_pullup** or **mark_weak_pullup** commands.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
erase_weak_pullup(2)
mark_weak_pullup(2)
match_topology(2)
topo_weak_pullup_n_length(3)
topo_weak_pullup_n_width(3)
topo_weak_pullup_p_width(3)
```

topo_weak_pullup_p_width

Controls the width or width to length ratio requirement for recognizing PMOS weak pullup transistors.

TYPE

float

DEFAULT

0.0

DESCRIPTION

NanoTime recognizes NMOS and PMOS weak pullup transistors. A PMOS weak pullup transistor has its source connected to Vdd, its gate tied to logic 0, and its drain connected to the pullup node. The transistor is turned on, but it is weak enough to allow a stronger transistor to pull the node down to a low voltage. NanoTime ignores weak pullup transistors during logic propagation.

To determine whether a PMOS transistor is weak enough to be considered a weak pullup, NanoTime considers the following variables:

topo_weak_pullup_p_length
topo_weak_pullup_p_width

If both variables are set to zero (the default), automatic recognition of PMOS weak pullup transistors is disabled. (However, you can still identify weak pullup transistors with the **mark_weak_pullup** command.)

If both variables are set to nonzero values, NanoTime compares the width to length ratio of each PMOS transistor connected in a pullup configuration with the width to length ratio specified by the two variables. If the transistor is weaker (has a lower width to length ratio), it is considered a weak pullup. Otherwise, it is not considered a weak pullup and NanoTime considers the drive strength during logic propagation.

If only the length variable is set to a nonzero value, a PMOS transistor connected in a pullup configuration is considered a weak pullup if its length is more than the variable-specified value.

Similarly, if only the width variable is set to a nonzero value, a PMOS transistor connected in a pullup configuration is considered a weak pullup if its width is less than the variable-specified value.

You can override the weak or not-weak pullup determination for any transistor by using the **erase_weak_pullup** or **mark_weak_pullup** commands.

This variable affects topology recognition. If you set it after the **match_topology** command, you should use the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

SEE ALSO

```
erase_weak_pullup(2)
mark_weak_pullup(2)
match_topology(2)
topo_weak_pullup_n_length(3)
topo_weak_pullup_n_width(3)
topo_weak_pullup_p_length(3)
```

trace_disable_switching_net_logic_check

Specifies whether to consider switching nets when doing false path logic checking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime does logical false path checking during path tracing. For nets on the path of interest, the logic checking is performed based on the switching direction. If there is a conflict between the switching direction and other logic settings through associated logical constraints, the path is pruned as a false path.

If this variable is set to true, NanoTime does not consider any switching nets when doing false path logic checking.

SEE ALSO

`trace_paths(2)`

trace_enable_modified_clock_waveforms

Allows enable signals at clock gates to modify or chop the clock waveforms generated at the clock gate output.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, clock waveforms are only propagated from the clock pin of a clock-gate topology to the output of the clock gate. If you mark the clock gate as transparent and set this variable to **true**, the gating signal can truncate or chop the opening edge of the clock pulse coming through the gate. Additionally, if the **trace_transparent_clock_gate_propagate_closing_edge** variable is set to **true**, the closing edge of the clock pulse can also be truncated or chopped.

For this variable to have any effect on the opening edge of the clock pulse, the max enabling arrival of the clock gate enable signal must arrive after the start of the clock pulse for opening edge truncation. In other words, a positive enable pin rising arrival would arrive after the rising edge of the clock signal. This would typically be a clock gate setup violation if the clock gate remained nontransparent.

For this variable to have any effect on the closing edge of the clock pulse, the min enabling arrival of the clock gate enable signal must arrive before the end of the clock pulse for closing edge truncation. In other words, a positive enable pin falling arrival would arrive before the falling edge of the clock signal. This is always a hold violation at the clock gate input.

SEE ALSO

```
trace_transparent_clock_gate_propagate_closing_edge(3)
remove_non_transparent(2)
report_clock_arrivals(2)
```

trace_gated_clock_error_recovery

Specifies whether to adjust the arrival time and continue path tracing when a setup violation occurs at a transparent gated clock.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, when NanoTime encounters a setup violation at a transparent gated clock, the tool stops path tracing. If you want to continue path tracing when a setup error occurs, set this variable to **true**. The tool adjusts the data arrival time back to zero-slack value and continues tracing the path forward.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

trace_latch_error_recovery

Specifies whether to adjust the arrival time and continue path tracing when a setup violation occurs at a transparent latch.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, when NanoTime encounters a setup violation at a transparent latch, the tool adjusts the data arrival time back to zero-slack value and continues tracing the path forward. The **report_paths** command reports this condition as a "latch error recovery" adjustment in a detailed path report. This behavior provides information on the path timing downstream from the location of the setup error.

If you would rather stop path tracing when a setup error occurs, set this variable to **false**. This behavior saves runtime but provides no information about the path downstream from the setup error.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

trace_search_depth_limit

Specifies the maximum number of successive timing stages that can be traced within a given path.

TYPE

integer

DEFAULT

1000000

DESCRIPTION

This variable specifies the largest number of successive timing stages that can be traced within a given path. A timing stage is typically one channel-connected block. During path tracing, if NanoTime encounters this number of timing stages in a path, it stops tracing for that path and issues a warning message.

This variable is set by default to 1000000, which in practical terms means no limit to the number of stages per path. Set this variable to a lower number for debugging purposes when you suspect that paths are not being correctly traced.

SEE ALSO

```
report_paths(2)
trace_paths(2)
trace_transparency_depth_limit(3)
```

trace_show_stack_period

Enables periodic reporting of the state of path tracing and specifies the number of minutes between periodic reports.

TYPE

integer

DEFAULT

0

DESCRIPTION

This variable specifies the number of minutes between periodic progress reports generated by the **trace_paths** command. Each report shows the current state of path tracing. If the variable is set to 0 (the default), no periodic reports are generated.

SEE ALSO

`trace_paths(2)`

trace_through_inputs

Specifies whether to continue path tracing when an input port is encountered in a path.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime stops path tracing every time it encounters an input port at an intermediate point in a path. To have NanoTime continue tracing a path when it encounters an input port, set this variable to **true**.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

trace_through_outputs

Specifies whether to continue path tracing when an output port is encountered in a path.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, NanoTime stops path tracing when it encounters an output port at an intermediate point in a path. To have NanoTime continue tracing a path when it encounters an output port, set this variable to **true**.

SEE ALSO

`report_paths(2)`
`trace_paths(2)`

trace_transparency_depth_closing_edge

Determines how NanoTime handles the last latch found in a path when the limit set in the **trace_transparency_depth_limit** variable is reached.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **trace_transparency_depth_closing_edge** variable determines how NanoTime handles the last latch found in a path when the limit set in the **trace_transparency_depth_limit** variable is reached.

The **trace_transparency_depth_limit** variable specifies the largest number of latches that can be traversed transparently for any single path. During tracing of a path, if NanoTime encounters this number of transparent latches, the last latch is assumed to be nontransparent if the **trace_transparency_depth_closing_edge** variable is set to **false** (the default). If the **trace_transparency_depth_closing_edge** variable is set to **true**, NanoTime assumes that the last latch can be operated in transparent mode and therefore is checked on the closing edge of its transparency window.

SEE ALSO

`trace_transparency_depth_limit(3)`

trace_transparency_depth_limit

Specifies the maximum number of latches that can be traversed transparently within any single path.

TYPE

integer

DEFAULT

1000000

DESCRIPTION

The **trace_transparency_depth_limit** variable specifies the largest number of latches that can be traversed transparently for any single path. During tracing of a path, if NanoTime encounters this number of transparent latches, the last latch is assumed to be nontransparent. If the **trace_transparency_depth_closing_edge** variable is set to **true**, the tool assumes that the last latch is transparent and the check occurs on the closing edge of that latch's transparency window.

The default of the **trace_transparency_depth_limit** variable is 1000000, which in practical terms means no limit to the number of transparent latches traced per path. Set this variable to a lower number to reduce runtime when you know that all paths contain no more than that number of transparent latches.

SEE ALSO

```
report_paths(2)
trace_paths(2)
trace_search_depth_limit(3)
trace_transparency_depth_closing_edge(3)
```

trace_transparent_clock_gate_propagate_closir

Determines how NanoTime handles the closing edge of a gated clock pulse with transparent behavior.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, gated clocks are nontransparent, so the path to the enable or gating pin of the clock gate ends at that point.

If the gated clock is changed to transparent by the **remove_non_transparent** command, the gating signal is allowed to propagate through the gating circuit as it chops the clock pulse.

By default, only the opening edge of the pulse is allowed to propagate. However, if the **trace_transparent_clock_gate_propagate_closing_edge** variable is set to **true**, the chopped closing edge of the clock pulse is also allowed to propagate.

SEE ALSO

`remove_non_transparent(2)`

trace_transparent_inverting_loops

Specifies whether to consider the inversion status of a signal being traced through a clocked loop path.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, during tracing of a clocked loop path, NanoTime does not consider the inversion or noninversion of the signal in the path. If a path returns to the same latch from which it originated, it is considered a loop path and is reported as such.

If this variable is set to **true**, NanoTime considers the switching direction in the path. In that case, the loop is considered completed after one time or two times around the loop, depending on the switching direction after one full traversal through the loop. If, after tracing through the loop one time, the signal is inverted with respect to the start of the loop, tracing continues beyond one full loop and continues until two times around the loop. If the signal is not inverted after one full loop, the loop is considered completed at that point.

Here is an example of an inverting loop path traced with the variable set to **true**:

L1 (rising) -> L2 (falling) -> L3 (rising) -> L1 (falling) -> L2 (rising) -> L3 (falling) -> L1 (rising)

With the variable set to **false**, path tracing stops upon completion of one loop, regardless of the inversion:

L1 (rising) -> L2 (falling) -> L3 (rising) -> L1 (falling)

This variable has an effect only if the **trace_transparent_loop_checking** variable is set to **true**.

SEE ALSO

trace_transparent_loop_checking(3)

trace_transparent_loop_checking

Specifies whether to perform transparent loop path checking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, NanoTime looks for and reports the condition where a path going through a loop of transparent latches takes more time than the number of cycles allowed for that set of latches. When the variable is **false** (the default), there is no transparent loop checking.

NanoTime detects a transparent loop path when a path starts from a latch, transparently traverses one or more latches, and returns to the first latch. When the **trace_transparent_loop_checking** variable is set to **true**, NanoTime computes the loop path delay up to the loop node.

SEE ALSO

```
report_paths(2)
trace_paths(2)
trace_search_depth_limit(3)
trace_transparency_depth_closing_edge(3)
trace_transparent_inverting_loops(3)
```

transistor_stack_bidirectional_limit

Specifies the maximum number of bidirectional transistors considered for processing in a delay arc.

TYPE

integer

DEFAULT

2

DESCRIPTION

This variable specifies a limit on the number of bidirectional transistors through which NanoTime searches when the tool tries to find a path from power or ground to a signal node. Setting a lower limit can save CPU time. However, NanoTime cannot process transistor stacks containing more than the specified number of bidirectional transistors.

SEE ALSO

```
set_transistor_direction(2)
transistor_stack_height_limit(3)
```

transistor_stack_height_limit

Specifies the maximum number of transistors in a stack considered for finding stage delay arcs.

TYPE

integer

DEFAULT

10

DESCRIPTION

This variable specifies the limit on the number of transistors in a channel-connected stack through which NanoTime searches when the tool tries to find a path from power or ground to a signal node. Setting a lower limit can save CPU time. However, NanoTime cannot process transistor stacks containing more than the specified number of transistors.

SEE ALSO

`set_transistor_direction(2)`
`transistor_stack_bidirectional_limit(3)`

weak_gnd_ic_multiplier_limit

Specifies the highest initial condition voltage allowed on "weak GND" nets within a PMOS stack as a ratio of the supply voltage.

TYPE

float

DEFAULT

1.0

DESCRIPTION

Consider a stack of PMOS devices that are channel-connected to the supply rail and to the ground rail (usually through NMOS devices in the same CCB). When a PMOS device in the stack is off, the voltages at the internal nodes of the stack are limited by the threshold voltage (Vt) of an on device. In very low voltage designs, this can lead to initial condition voltages that approach the opposite rail voltage (in this case, the supply rail).

This variable specifies the maximum value of the initial condition voltage that can be assigned to an internal node of a PMOS stack. Valid values are 0.0 (ground) to 1.0 (the supply voltage).

The **weak_gnd_ic_multiplier_limit** variable does not apply to initial conditions for devices such as pass transistors that are not channel-connected to a rail.

SEE ALSO

`weak_vdd_ic_multiplier_limit(3)`

```
set_initial_condition_adjustment(2)
```

weak_vdd_ic_multiplier_limit

Specifies the lowest initial condition voltage allowed on "weak VDD" nets within an NMOS stack as a ratio of the supply voltage.

TYPE

float

DEFAULT

0.0

DESCRIPTION

NanoTime sets initial condition values on internal nodes of a channel-connected block (CCB) before simulation.

Consider a stack of NMOS devices that are channel-connected to the ground rail and to the supply rail (usually through PMOS devices in the same CCB). When an NMOS device in the stack is off, the voltages at the internal nodes of the stack are limited by the threshold voltage (Vt) of an on device. In very low voltage designs, this can lead to initial condition voltages that approach the opposite rail voltage (in this case, ground).

This variable specifies the minimum value of the initial condition voltage that can be assigned to an internal node of an NMOS stack. Valid values are 0.0 (ground) to 1.0 (the supply voltage).

The **weak_vdd_ic_multiplier_limit** variable does not apply to initial conditions for devices such as pass transistors that are not channel-connected to a rail.

SEE ALSO

```
weak_gnd_ic_multiplier_limit(3)
set_initial_condition_adjustment(2)
```

write_spice_sim_cmd

Specifies the command used to launch simulations of SPICE decks.

TYPE

string

DEFAULT

hspice -i <input_deck> -o <listing_file>

DESCRIPTION

The **write_spice_sim_cmd** variable specifies the command to launch simulations in your computing environment. The default is to use the HSPICE simulator.

This variable has an effect only if you use the **-simulate** option with the **write_spice** command. NanoTime uses the information in the **write_spice_sim_cmd** variable to launch simulations of the SPICE decks that were created by the **write_spice** command. Simulations are launched in parallel if distributed processing is enabled in your computing environment; otherwise, they are executed sequentially.

The **<input_deck>** and **<listing_file>** tags are syntax terms and must be written *exactly* as specified, including the angle brackets. Do not change the wording inside the angle brackets. The NanoTime tool replaces the tags with values as follows:

- The **<input_deck>** tag is replaced with one of the SPICE deck file names created by the **write_spice -simulate** command. For example, a file name might be abc_00003.sp.
- The **<listing_file>** tag is replaced with the root name of the SPICE deck file name. For example, the root name is abc_00003 for file abc_00003.sp.

NanoTime obtains the SPICE deck root name and file name from the argument that you supply for the **-output** option of the **write_spice** command. The tool creates file names for multiple SPICE decks by appending an index number to the root name. The original file extension that you provide is used as the file

extension for all derived SPICE deck file names.

You can use **write_spice_sim_pre_processing** and **write_spice_sim_post_processing** to process your SPICE decks and simulation results, respectively.

If you do not want to run any simulations on the SPICE decks, but you want to process the SPICE deck files with your own script, set the **write_spice_sim_cmd** variable to an empty string and specify your script name in the **write_spice_sim_pre_processing** variable.

If the simulator is the HSPICE simulator, the **distributed_processing_number_hspice_license** variable specifies the upper limit for the number of consumed HSPICE licenses. HSPICE Advanced Client/Server mode is used.

For all simulators, the **write_spice_sim_max** variable is the maximum number of simulations to be run. This limit also applies to preprocessing or postprocessing script usage, even if no simulations are performed.

EXAMPLES

The following command specifies to use the FineSim simulator instead of the default choice of the HSPICE simulator.

```
nt_shell> set write_spice_sim_cmd "finesim <input_deck> -output <listing_file> -w "
```

The following commands specify that no simulation should be automatically performed, but script pre_calc.tcl should be executed on each of the SPICE deck files.

```
nt_shell> set write_spice_sim_pre_processing "pre_calc.tcl <input_deck>"
nt_shell> set write_spice_sim_cmd ""
```

The following commands specify to use the FineSim simulator and use get_path_delay.pl to process the simulation results.

```
nt_shell> set write_spice_sim_cmd "finesim <input_deck> -output <listing_file> -w "
nt_shell> set write_spice_sim_post_processing "get_path_delay.pl <listing_file>.mt0"
```

SEE ALSO

```
write_spice(2)
write_spice_sim_pre_processing(3)
write_spice_sim_post_processing(3)
write_spice_sim_max(3)
write_spice_sim_max_decks_per_task(3)
distributed_processing_number_hspice_license(3)
```

write_spice_sim_max

Specifies the maximum number of SPICE simulations to be launched.

TYPE

integer

DEFAULT

250000

DESCRIPTION

The **write_spice_sim_max** variable specifies the maximum number of simulations to be launched by the **write_spice -simulate** command. This limit also applies to preprocessing or postprocessing script usage, even if no simulations are performed. Simulations are launched in parallel if distributed processing is enabled in your computing environment; otherwise, they are executed sequentially.

SEE ALSO

```
write_spice(2)
write_spice_sim_cmd(3)
write_spice_sim_max_decks_per_task(3)
write_spice_sim_pre_processing(3)
write_spice_sim_post_processing(3)
```

write_spice_sim_max_decks_per_task

Specifies the maximum number of SPICE decks to group into one task for simulation.

TYPE

integer

DEFAULT

20

DESCRIPTION

The **write_spice_sim_max_decks_per_task** variable specifies the maximum number of SPICE decks packed in one task by the **write_spice -simulate** command. The simulation of each SPICE deck includes preprocessing, simulation, and postprocessing. Simulations are launched in parallel if distributed processing is enabled in your computing environment; otherwise, they are executed sequentially.

SEE ALSO

```
write_spice(2)
write_spice_sim_cmd(3)
write_spice_sim_max(3)
write_spice_sim_pre_processing(3)
write_spice_sim_post_processing(3)
```

write_spice_sim_post_processing

Specifies an optional script file name to be executed on a SPICE deck after simulation.

TYPE

string

DEFAULT

""

DESCRIPTION

The **write_spice -simulate** command, when applied to a list of timing paths, creates multiple SPICE decks and automatically launches simulations of those SPICE decks.

You can optionally use the **write_spice_sim_pre_processing** variable to provide a script file name or an operating system command to be executed on each SPICE deck before simulation. For example, you might want to search for and rename specific elements in a SPICE deck before simulation.

Similarly, you can use the **write_spice_sim_post_processing** variable to provide a script file name or an operating system command to be executed immediately after a simulation run is completed. For example, you might want to extract values from one or more of the results files to create a customized report.

If you set either or both of these variables, the scripts are executed even if you disable simulation by setting the **write_spice_sim_cmd** variable to an empty string.

For both the **write_spice_sim_pre_processing** and **write_spice_sim_post_processing** variables, you can use the **<input_deck>** and **<listing_file>** tags to indicate, respectively, the name of the SPICE deck file or the root name of the file (the name without the file extension). These two tags are syntax terms and must be written *exactly* as specified, including the angle brackets. Do not change the wording inside the angle brackets.

Simulations are launched in parallel if distributed processing is enabled in your computing environment. The preprocessing and postprocessing scripts are considered to be part of the simulation run and are therefore also executed in parallel. In other words, the complete analysis of file abc_00023.sp might include a preprocessing step, a simulation run, and a postprocessing step. Multiple instances of this three-step analysis flow are launched in parallel if distributed processing is enabled. Otherwise, each three-step flow is executed sequentially.

EXAMPLES

The following command specifies to use script post_calc.tcl to process the measurement output file from HSPICE simulation of the SPICE deck files.

```
nt_shell> set write_spice_sim_post_processing "post_calc.tcl <listing_file>.mt0"
```

SEE ALSO

write_spice(2)
write_spice_sim_pre_processing(3)
write_spice_sim_cmd(3)
write_spice_sim_max(3)
write_spice_sim_max_decks_per_task(3)

write_spice_sim_pre_processing

Specifies an optional script file name to be executed on a SPICE deck before simulation.

TYPE

string

DEFAULT

""

DESCRIPTION

The **write_spice -simulate** command, when applied to a list of timing paths, creates multiple SPICE decks and automatically launches simulations of those SPICE decks.

You can optionally use the **write_spice_sim_pre_processing** variable to provide a script file name or an operating system command to be executed on each SPICE deck before simulation. For example, you might want to search for and rename specific elements in a SPICE deck before simulation.

Similarly, you can use the **write_spice_sim_post_processing** variable to provide a script file name or an operating system command to be executed immediately after a simulation run is completed. For example, you might want to extract values from one or more of the results files to create a customized report.

If you set either or both of these variables, the scripts are executed even if you disable simulation by setting the **write_spice_sim_cmd** variable to an empty string.

For both the **write_spice_sim_pre_processing** and **write_spice_sim_post_processing** variables, you can use the **<input_deck>** and **<listing_file>** tags to indicate, respectively, the name of the SPICE deck file or the root name of the file (the name without the file extension). These two tags are syntax terms and must be written *exactly* as specified, including the angle brackets. Do not change the wording inside the angle brackets.

Simulations are launched in parallel if distributed processing is enabled in your computing environment. The preprocessing and postprocessing scripts are considered to be part of the simulation run and are therefore also executed in parallel. In other words, the complete analysis of file abc_00023.sp might include a preprocessing step, a simulation run, and a postprocessing step. Multiple instances of this three-step analysis flow are launched in parallel if distributed processing is enabled. Otherwise, each three-step flow is executed sequentially.

EXAMPLES

The following command specifies to use script pre_proc.sh to process the SPICE deck files.

```
nt_shell> set write_spice_pre_processing "pre_proc.sh <input_deck>"
```

The following command specifies to use the cp UNIX command to back up the SPICE deck files.

```
nt_shell> set write_spice_sim_pre_processing "cp <input_deck> /backup_folder/<listing_file>_bkup.sp
```

SEE ALSO

write_spice(2)
write_spice_sim_post_processing(3)
write_spice_sim_cmd(3)
write_spice_sim_max(3)
write_spice_sim_max_decks_per_task(3)

write_spice_sim_validate

Specifies the command to validate the results generated by preprocessing, simulation, and/or postprocessing.

TYPE

string

DEFAULT

test -f <listing_file>.mt0

DESCRIPTION

The **write_spice_sim_validate** variable specifies the command to validate the results generated by preprocessing, postprocessing, and/or simulation .

This variable has an effect only if you use the **-simulate** option with the **write_spice** command. The non-zero return value of the command indicates the failure of the results validation and NnaoTime will attempt to rerun the task.

The **<input_deck>** and **<listing_file>** tags are syntax terms and must be written *exactly* as specified, including the angle brackets. Do not change the wording inside the angle brackets. The NanoTime tool replaces the tags with values as follows:

- The **<input_deck>** tag is replaced with one of the SPICE deck file names created by the **write_spice -simulate** command. For example, a file name might be abc_00003.sp.
- The **<listing_file>** tag is replaced with the root name of the SPICE deck file name. For example, the root name is abc_00003 for file abc_00003.sp.

NanoTime obtains the SPICE deck root name and file name from the argument that you supply for the **-output** option of the **write_spice** command. The tool creates file names for multiple SPICE decks by appending an index number to the root name. The original file extension that you provide is used as the file

extension for all derived SPICE deck file names.

EXAMPLES

The following command specifies to check two files expected to be generated after the HSPICE simulation.

```
nt_shell> set write_spice_sim_validate "test -f <listing_file>.mt0 && test -f <listing_file>.lis"
```

SEE ALSO

write_spice(2)
write_spice_sim_pre_processing(3)
write_spice_sim_post_processing(3)