NanoTime
Tool Commands

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

add_to_collection

Adds objects to a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection add_to_collection
   [-unique]
   base_collection
   object_spec
```

Data Types

base_collection	collection
object_spec	list

ARGUMENTS

-unique

Causes duplicate objects to be removed from the resulting collection. By default, duplicate objects are not removed.

base_collection

Specifies the base collection to which objects are added. The base collection can be the empty collection (empty string).

object_spec

Specifies a list of named objects or collections to add. If the name matches an existing collection, the collection is added. Otherwise, objects in the list that match the object class of the base collection are added.

DESCRIPTION

The **add_to_collection** command adds elements to an existing collection. The result is a new collection

representing the objects in the *object_spec* added to the objects in the base collection. If the *object_spec* is empty, the result is a copy of the base collection.

By default, elements that exist in both the base collection and the *object_spec* are duplicated in the resulting collection. Use the **-unique** option to remove duplicates.

For homogeneous base collections:

All collections in the *object_spec*, whether homogeneous or heterogeneous, are added to the base collection.

All individual objects in the *object_spec* that match the object class of the base collection are added to the base collection.

For heterogeneous base collections:

All collections in the *object_spec*, whether homogeneous or heterogeneous, are added to the base collection.

Individual objects in the *object_spec* cannot be added to the base collection; NanoTime issues a warning message.

For empty base collections:

Adding two or more empty collections yields the empty collection.

There must be at least one collection in a non-empty *object_spec*; its position in the list does not matter. The first collection in the *object_spec* becomes the base collection for the new collection. If it is a homogeneous collection, it sets the object class for the new collection. If it is a heterogeneous collection, there is no object class for the new collection.

Individual objects in the *object_spec* are added to the collection only if they match the object class of a new homogeneous base collection. Their positions in the list do not matter. NanoTime issues a warning message if there are objects in the *object_spec* that do not match the object class of the new base collection or if the new base collection is heterogeneous.

All other collections in the *object_spec*, whether homogeneous or heterogeneous, are added to the new base collection. The object classes in these additional collections do not trigger any additional matching of individual objects in the *object_spec*.

COMMAND PHASING

The **add_to_collection** command can be executed at any time.

EXAMPLES

The following set of commands gets all ports beginning with "mode," then adds the CLOCK port to the collection.

```
nt_shell> set xports [get_ports mode*]
```

```
{"mode[0]", "mode[1]", "mode[2]"}
nt_shell> add_to_collection $xports [get_ports CLOCK]
{"mode[0]", "mode[1]", "mode[2]", "CLOCK"}
```

The following set of commands adds the cell u1 to a collection containing the SCANOUT port.

```
nt_shell> set sp [get_ports SCANOUT]
{"SCANOUT"}
nt_shell> set het [get_cells u1]
{"u1"}
nt_shell> add_to_collection $sp $het
{"SCANOUT", "u1"}
```

The following examples show how the **add_to_collection** command behaves when the base collection is empty. The variables have the same values as in the previous example.

```
nt_shell> sizeof_collection [add_to_collection "" ""]
0

nt_shell> set A [add_to_collection "" [list a b c]]
Error: At least one collection required for argument 'object_spec'
 to add_to_collection when the 'collection' argument is empty (SEL-014)

nt_shell> add_to_collection "" [list a b c $het $sp]
Warning: Ignored all implicit elements in argument 'object_spec'
 to add_to_collection because the class of the base collection
 could not be determined (SEL-015)
{"SCANOUT", "u1", "SCANOUT"}
```

SEE ALSO

```
collections(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
```

alias

Creates a pseudo-command that expands to one or more words, or lists current alias definitions.

SYNTAX

```
string alias
[name]
[def]
```

Data Types

```
name      string
def       string
```

ARGUMENTS

name

Specifies a name of the alias to define or display. The name must begin with a letter, and can contain letters, underscores, and numbers.

def

Expands the alias. That is, the replacement text for the alias name.

DESCRIPTION

The **alias** command defines or displays command aliases. With no arguments, the **alias** command displays all currently defined aliases and their expansions. With a single argument, the **alias** command displays the expansion for the given alias name. With more than one argument, an alias is created that is named by the first argument and expanding to the remaining arguments.

You cannot create an alias using the name of any existing command or procedure. Thus, you cannot use **alias** to redefine existing commands.

Aliases can refer to other aliases.

Aliases are only expanded when they are the first word in a command.

EXAMPLES

Although commands can be abbreviated, sometimes there is a conflict with another command. The following example shows how to use **alias** to get around the conflict:

```
prompt> alias q quit
```

The following example shows how to use **alias** to create a shortcut for commonly-used command invocations:

```
prompt> alias include {source -echo -verbose}

prompt> alias rt100 {report_timing -max_paths 100}
```

After the previous commands, the command **include script.tcl** is replaced with **source -echo -verbose script.tcl** before the command is interpreted.

The following examples show how to display aliases using **alias**. Note that when displaying all aliases, they are in alphabetical order.

```
prompt> alias rt100
rt100  report_timing -max_paths 100

prompt> alias
include  source -echo -verbose
q  quit
rt100  report_timing -max_paths 100
```

SEE ALSO

unalias(2)

all_clocks

Creates a collection of all clocks in the current design. You can assign these clocks to a variable or pass them into another command.

SYNTAX

```
collection all_clocks
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **all_clocks** command creates a collection of all clock objects in the current design. If you do not define any clocks, the empty collection (empty string) is returned.

If you want to reference only certain clocks, use the **get_clocks** command to create a collection of clocks that match a specific pattern or are specified by filter criteria.

COMMAND PHASING

The **all_clocks** command can be executed any time after the **link_design** command.

EXAMPLES

The following example applies the **set_propagated_clock** command to all clocks in the design.

```
nt_shell> set_propagated_clock [all_clocks]
```

SEE ALSO

collections(2)
create_clock(2)
get_clocks(2)
set_propagated_clock(2)

all_inputs

Creates a collection of all input ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_inputs
   [-level_sensitive]
   [-edge_triggered]
   [-clock clock_name]
```

Data Types

clock_name	list

ARGUMENTS

-level_sensitive

Only considers ports with level-sensitive input delay, as specified by the **set_input_delay -level_sensitive** command.

-edge_triggered

Only considers ports with edge-triggered input delay, as specified by the **set_input_delay** command without the **-level_sensitive** option.

-clock *clock_name*

Only considers ports with input delay relative to a specific clock. This can be the name of a clock, or a collection containing a clock.

DESCRIPTION

The **all_inputs** command creates a collection of all input or inout ports in the current design. You can limit

the contents of the collection by specifying the type of input delay that must be present on a port.

If you want to specify only certain ports, use the **get_ports** command to create a collection of ports that match a specific pattern or are specified by filter criteria.

When issued from the command prompt, the **all_inputs** command behaves as if the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this number using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the man page for the **collections** topic.

COMMAND PHASING

The **all_inputs** command can be executed any time after the **link_design** command.

EXAMPLES

The following command sets the pin capacitance for all input ports.

```
nt_shell> set_load 1.2 [all_inputs]
```

SEE ALSO

```
collections(2)
get_ports(2)
report_port(2)
query_objects(2)
set_driving_cell(2)
set_input_delay(2)
collection_result_display_limit(3)
```

all_instances

Creates a collection of all instances of a specific design (or library cell) in the current design, relative to the current instance. You can assign the resulting collection of cells to a variable or pass it into another command.

SYNTAX

```
collection all_instances
   [-hierarchy]
   object_spec
```

Data Types

object_spec	list

ARGUMENTS

-hierarchy

Searches for instances in all levels of hierarchy below the current instance. By default, only instances from the current level of hierarchy are considered.

object_spec

Specifies the target design or library cell. This can be a design collection, a library cell collection, or a name.

DESCRIPTION

The **all_instances** command creates a collection of cells that are instances of a design or library cell. The search for instances is made relative to the current instance within the current design. By default, the **all_instances** command considers only instances at the current level of the hierarchy. If you use the **-hierarchy** option, the search continues throughout the hierarchy.

The *object_spec* can be a simple name. In this case, any instance of a design or library cell with that

name is a match. Alternatively, the *object_spec* can be a collection of exactly one design or one library cell. Any other collection results in an error message. Using the collection can help focus the search for instances of specific designs or library cells.

When issued from the command prompt, the **all_instances** command behaves as if the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this number using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **all_instances** command can be executed any time after the **link_design** command.

EXAMPLES

The following example uses the **all_instances** command to get the instances of the design named "low" in the current level of hierarchy.

```
nt_shell> all_instances low
{"U1", "U3"}
```

The following example uses the **-hierarchy** option to display instances of the design named "low" across multiple levels of hierarchy.

```
nt_shell> query_objects [all_instances low -hierarchy]
{"U1", "U2/U1", "U3"}
```

SEE ALSO

collections(2)
current_design(2)
current_instance(2)
get_designs(2)
get_lib_cells(2)
get_cells(2)
list_designs(2)
query_objects(2)
collection_result_display_limit(3)

all_outputs

Creates a collection of all output ports in the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection all_outputs
   [-level_sensitive]
   [-edge_triggered]
   [-clock clock_name]
```

Data Types

```
   clock_name            list
```

ARGUMENTS

-level_sensitive

Only considers ports with level-sensitive output delay, as specified by the **set_output_delay -level_sensitive** command.

-edge_triggered

Only considers ports with edge-triggered output delay, as specified by the **set_output_delay** command without the **-level_sensitive** option.

-clock *clock_name*

Only considers ports with output delay relative to a specific clock. This can be the name of a clock, or a collection containing a clock.

DESCRIPTION

The **all_outputs** command creates a collection of all output or inout ports in the current design. You can

limit the contents of the collection by specifying the type of output delay that must be present on a port.

If you want to specify only certain ports, use the **get_ports** command to create a collection of ports that match a specific pattern or are specified by filter criteria.

When issued from the command prompt, the **all_outputs** command behaves as if the **query_objects** command had been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this number using the **collection_result_display_limit** variable.

For information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **all_outputs** command can be executed any time after the **link_design** command.

EXAMPLES

The following command sets the pin capacitance for all output ports.

```
nt_shell> set_load 4.56 [all_outputs]
```

The following command shows the names all of the output ports with output delay relative to PHI1.

```
nt_shell> all_outputs -clock PHI1
```

SEE ALSO

```
collections(2)
get_ports(2)
report_port(2)
query_objects(2)
set_output_delay(2)
collection_result_display_limit(3)
```

all_simulations

Creates a collection of all dynamic delay simulation objects in the design.

SYNTAX

```
collection all_simulations
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **all_simulations** command creates a collection of all dynamic delay simulation (DDS) objects in the design. A DDS simulation object is a region in the design specified with the **mark_simulation** command and recognized by the **check_design** command. You must have a NanoTime Ultra license to use dynamic delay simulation.

You can set a variable to the collection and then perform operations on the collection using commands such as the **report_simulation** and **set_simulation_attributes** commands.

To create a collection of a subset of the simulation objects in the design rather than all of them, use the **get_simulations** command.

COMMAND PHASING

The **all_simulations** command must be executed after the **check_design** command but before the **trace_paths** command.

EXAMPLES

The following command generates a report on all simulation objects in the design.

```
nt_shell>  report_simulation [all_simulations]

Summary of Simulations

                            Num     Num     Num   Num
Outputs                     Outputs Inputs  Nets  TX
--------------------------- ------- ------- ----- -----
 ...
```

The following commands first set the variable **mysims** to the collection of all simulation objects in the design, then generate a report on those simulation objects.

```
nt_shell> set mysims [all_simulations]
Information: Defining new variable 'mysims'. (CMD-041)
_sel7
nt_shell> report_simulation $mysim
Summary of Simulations
                            Num     Num     Num   Num
Outputs                     Outputs Inputs  Nets  TX
--------------------------- ------- ------- ----- -----
 ...
```

SEE ALSO

```
get_simulations(2)
mark_simulation(2)
report_simulation(2)
set_simulation_attributes(2)
```

append_to_collection

Adds objects to a collection.

SYNTAX

```
collection append_to_collection
   [-unique]
   base_collection
   object_spec
```

Data Types

base_collection	collection
object_spec	list

ARGUMENTS

-unique

Causes duplicate objects to be removed from the resulting collection. By default, duplicate objects are not removed.

base_collection

Specifies the base collection to which objects are added. The objects matching the *object_spec* list are added into the collection referenced by this argument.

object_spec

Specifies a list of named objects or collections to add.

DESCRIPTION

The **append_to_collection** command allows you to add elements to a collection. This command treats the *base_collection* argument as a collection, and appends all of the elements in the *object_spec* list to that collection. If the collection referenced by *base_collection* does not exist, it is created as a collection

with elements from the *object_spec* list as its value. If the collection referenced by *base_collection* does exist, and it does not contain a collection, NanoTime issues an error message.

The result of the command is the collection which was initially referenced by *base_collection*, or the collection created if the variable did not exist.

The **append_to_collection** command provides the same semantics as a common use of the **add_to_collection** command, but with added flexibility. The following two commands have the same result:

```
set collection_2 [add_to_collection $collection_2 $objs]

append_to_collection collection_2 $objs
```

The **append_to_collection** command can be much more efficient than the **add_to_collection** command if you are building up a collection in a loop. The arguments of the command have the same restrictions as the **add_to_collection** command. See the man page for the **add_to_collection** command for more information about those restrictions.

COMMAND PHASING

The **append_to_collection** command can be executed at any time.

EXAMPLES

The following example shows how a collection can be built up with the **append_to_collection** command:

```
nt_shell> set xports
Error: can't read "xports": no such variable
        Use error_info for more info. (CMD-013)
nt_shell> append_to_collection xports [get_ports in*]
{"in0", "in1", "in2"}
nt_shell> append_to_collection xports CLOCK
{"in0", "in1", "in2", "CLOCK"}
nt_shell> append_to_collection xports [get_nets or*]
{"in0", "in1", "in2", "CLOCK", "or1", "or2"}
```

SEE ALSO

```
collections(2)
add_to_collection(2)
foreach_in_collection(2)
index_collection(2)
remove_from_collection(2)
sizeof_collection(2)
```

apropos

Searches the command database for a pattern.

SYNTAX

```
string apropos
   [-symbols_only]
   pattern
```

Data Types

```
   pattern      string
```

ARGUMENTS

-symbols_only

Searches only command and option names.

pattern

Searches for the specified *pattern*.

DESCRIPTION

The **apropos** command searches the command and option database for all commands that contain the specified *pattern*. The *pattern* argument can include the wildcard characters asterisk (*) and question mark (?). The search is case-sensitive. For each command that matches the search criteria, the command help is printed as though **help -verbose** was used with the command.

Whereas **help** looks only at command names, **apropos** looks at command names, the command one-line description, option names, and option value-help strings. The search can be restricted to only command and option names with the **-symbols_only** option.

When searching for dash options, do not include the leading dash. Search only for the name.

EXAMPLES

In the following example, assume that the **get_cells** and **get_designs** commands have the **-exact** option. Note that without the **-symbols_only** option, the first search picks up commands which have the string "exact" in the one-line description.

```
prompt> apropos exact
 get_cells            # Create a collection of cells
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match cell names against patterns)

 get_designs          # Create a collection of designs
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match design names against patterns)

 real_time            # Return the exact time of day

prompt> apropos exact -symbols_only
 get_cells            # Create a collection of cells
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match cell names against patterns)

 get_designs          # Create a collection of designs
   [-exact]               (Wildcards are considered as plain characters)
   patterns               (Match design names against patterns)
```

SEE ALSO

help(2)

characterize_context

Captures the timing context of a list of instances, allowing the constraints on those instances to be set during hierarchical analysis using timing models.

SYNTAX

```
status characterize_context
   [-max_only]
   [-min_only]
   [-full_path_enumeration]
   [-npaths number_of_paths]
   [-keep_paths_within time_value]
   [-pbsa]
   [-debug_paths]
   cell_list
```

Data Types

```
   number_of_paths        integer
   time_value             float
   cell_list              list
```

ARGUMENTS

-max_only

Considers only the maximum delay (longest) paths for path tracing and characterization.

-min_only

Considers only the minimum delay (shortest) paths for path tracing and characterization.

-full_path_enumeration

Traces all paths using full path enumeration, without pruning. Due to the large runtime requirements, this option is practical only when used with the **set_find_path** command to limit the scope of the design being considered for characterization.

-npaths *number_of_paths*

Specifies the maximum number of paths to save in the path database for each startpoint-endpoint pair. The default is 1. Specify a larger number to save more paths, but use a value that is small enough to

keep the runtime and memory usage reasonable. This option can only be used together with the **-keep_paths_within** option.

-keep_paths_within ***time_value***

For each startpoint-endpoint pair, keeps timing paths that have a delay within *time_value* of the worst path delay. By default, the **-keep_paths_within** option is set to 0.0, which means that NanoTime keeps only the single worst path per startpoint-endpoint pair. To keep more paths, set the **-keep_paths_within** option to a time value larger than 0.0 and set the **-npaths** option to a number larger than 1. This puts more timing information in the path database, but also increases the runtime and memory usage.

-pbsa

Invokes path-based slack adjustment, which adjusts the calculated path delays based on the lengths of the path segments traced in the timing check. For more information, see the man page for the **report_pbsa_calculation** command.

-debug_paths

Writes a file to the working directory that lists the specific worst-case timing paths in the design that are being used to generate the boundary constraints.

cell_list

The list of instances in the current design to characterize.

DESCRIPTION

The **characterize_context** command captures the timing context of instances of subdesigns from the chip-level timing environment. The timing context of an instance consists of the waveforms of the clocks affecting the instance, input arrival times, output required times, and timing exceptions. You can use this command together with the **write_context** command to export timing information for a block to the chip-level environment.

The **characterize_context** command is a path-tracing process like the **trace_paths** command. It can be used only after the **check_design** command has been run, and cannot be used after the **trace_paths** command.

If you have already run the **trace_paths** command, you must first reset the design using the **reset_design** command and restart the analysis from the design linking stage. After you run the **check_design** command, you can use the **characterize_context** command.

COMMAND PHASING

The **characterize_context** command can only be executed after the **check_design** command. After successful execution, the state is changed to "paths-traced."

EXAMPLES

The following example first derives the timing-related context information for instance I1, then writes the context information to a Synopsys Design Constraints (SDC) file called I1.ntsh in the current directory.

```
nt_shell> characterize_context {I1}

nt_shell> write_context -derive_file_name {I1}
```

SEE ALSO

```
extract_model(2)
report_pbsa_calculation(2)
trace_paths(2)
write_context(2)
```

check_design

Checks the design for proper structure.

SYNTAX

```
status check_design
   [-complete_with completion_type]
   [-message_level level]
```

Data Types

```
   completion_type       string
   level                 string
```

ARGUMENTS

-complete_with *completion_type*

Indicates that a net with partially annotated parasitics and missing segments is to be completed by inserting capacitances and resistances according to *completion_type*. Allowed values are **zero**, which completes the net by inserting very small, near-zero values; and **rc**, which completes the net by inserting fixed capacitances and resistances specified by the **parasitics_completion_capacitance** and **parasitics_completion_resistance** variables. This option is equivalent to the **complete_net_parasitics -complete_with** or **read_parasitics -complete_with** commands.

-message_level *level*

Specifies the amount of information reported in the terminal window. The allowed settings are **silent**, **normal** (the default), **verbose**, and **debug**.

DESCRIPTION

The **check_design** command checks the current design for proper structure and prepares the design for path tracing. In a NanoTime analysis session, this command must be executed after the **check_topology** command has been run and after all timing constraints have been set, but before the **trace_paths**

command. The general NanoTime flow is as follows:

```
1. link_design
2. (optional) match_topology
3. check_topology
4. check_design
5. trace_paths
```

Here are some examples of commands that set timing constraints and analysis conditions. If these commands are used, they must be run before the **check_design** command:

```
set_input_delay
set_output_delay
set_false_path
set_multicycle_path
set_no_same_phase_path
set_case_analysis
set_logic_constraint
set_timing_check
set_data_check
set_gated_clock_check
set_find_path
read_parasitics
```

The **check_design** command divides the design into units called channel-connected blocks (CCBs). Each CCB is a set of transistors whose sources and drains are connected together in a network, in series or in parallel, or both. NanoTime checks the CCBs and reports any problems in interpreting their configurations. You can correct such problems by using topology-marking commands such as the **mark_*** and **erase_*** commands, possibly in conjunction with the **foreach_match** command.

To restrict the scope of the design checked by the **check_design** command and subsequently traced by the **trace_paths** command, use the **set_find_path** command before the **check_design** command.

The **-message_level** option specifies the amount of information reported by the **check_design** command in the terminal window. The allowed settings are as follows:

```
silent: no messages except for warnings and errors
normal (the default): brief messages
verbose: statistics about CCBs
debug: information about individual CCBs
```

COMMAND PHASING

The **check_design** command can only be executed in the "topology-checked" state (after the **check_topology** command has been successfully executed). After successful execution, the state is changed to "design-checked."

EXAMPLES

The following is the display after a typical **check_design** command:

```
nt_shell> check_design
```

```
Pre-processing structural constraints...
Done pre-processing structural constraints
Creating new timing graph...
Analyzing transistor structures for timing graph...
Building timing graph...
Warning: Warning messages might appear here
Checking timing graph for consistency...
1
nt_shell>
```

The following **check_design** command provides statistical information on CCBs in the design:

```
nt_shell> check_design -message_level verbose
Pre-processing structural constraints...
 ...
Histogram of number CCBs by number of inputs nets
Maximum number of inputs nets for any CCB is 18
Total number of CCBs is 2245

Inputs  CCBs     %  Net in a CCB with this number of inputs nets
------  ---- -----  -------------------------------------------
     1  1387  61.8  Xlc23.or
     2   193   8.6  Xareg.Xreg0.X2.A
     4   512  22.8  Xareg.Xreg0.X3.A
     5    17   0.8  Xaddsub.Xadder.Xla0.Xlgen.XC1.X0.A
   ...   ...   ...   ...
```

SEE ALSO

```
check_topology(2)
match_topology(2)
set_find_path(2)
trace_paths(2)
complete_net_parasitics(2)
topo_latch_enable_logic_propagation(3)
```

check_topology

Checks for errors in topologies.

SYNTAX

```
status check_topology
   [-message_level level]
```

Data Types

```
   level          string
```

ARGUMENTS

-message_level *level*

Specifies the amount of information reported. The allowed settings are **silent**, **normal** (the default), **verbose**, and **debug**.

DESCRIPTION

The **check_topology** command signifies the end of the clock propagation and topology recognition phase in a NanoTime flow. The command checks for and reports errors in recognized topologies.

In a NanoTime session, the **check_topology** command must be used after the **link_design** command and before the **check_design** command. If the flow includes any **match_topology** commands, the **check_topology** command must occur after all **match_topology** commands. The general NanoTime flow is as follows:

```
1. link_design
2. (optional) match_topology
3. check_topology
4. check_design
5. trace_paths
```

Use the **check_topology** command after specifying all topology-related information for your design using the **mark_*** and **erase_*** commands, possibly in conjunction with the **foreach_match** command. In addition, the **match_topology** and **report_topology** commands can help you to learn about the topology of the design before you exit the topology definition phase.

If you use the **check_topology** command without first using a **match_topology** command, the **check_topology** command performs the same functions as the **match_topology** command before it proceeds with final topology error checking.

The **-message_level** option specifies the amount of information reported by the **check_design** command in the terminal window. The allowed settings are as follows:

```
silent: no messages except for warnings and errors
normal (the default): brief messages
verbose: statistics about CCBs
debug: information about individual CCBs
```

COMMAND PHASING

The **check_topology** command can only be executed in the "netlist-linked" state (after the **link_design** command has been successfully executed). After successful execution, the state is changed to "topology-checked."

EXAMPLES

The following command checks for topology errors.

```
nt_shell> check_topology
Starting structure identification...
Structure identification done.
1
```

SEE ALSO

```
check_design(2)
foreach_match(2)
match_topology(2)
report_topology(2)
```

collections

Describes the methodology for creating collections of objects and querying objects in the database.

ARGUMENTS

This is not a command, and it has no arguments. This man page provides a description of collections and the commands available to work with them.

DESCRIPTION

Synopsys tools build an internal database of the netlist and the attributes applied to it. This database consists of several classes of objects, including designs, libraries, ports, cells, nets, pins, and clocks. Most commands operate on these objects.

A collection is a group of objects exported to the Tcl user interface.

Collections have an internal representation (the objects) and, sometimes, a string representation. The string representation is generally used only for error messages.

A set of commands to create and manipulate collections is provided as an integral part of the user interface. The collection commands encompass two categories: several commands that create collections of objects for use by another command, and one command that queries objects for viewing. The result of a command that creates a collection is a Tcl object that can be passed to another command. For a query command, although the visible output looks like a list of objects (a list of object names is displayed), the result is an empty string.

An empty string "" is equivalent to the empty collection, that is, a collection with zero elements.

Homogeneous and Heterogeneous Collections

A homogeneous collection contains only one type of object. A heterogeneous collection contains more than one type of object. Commands that accept collections as arguments can accept either type of collection.

Lifetime of a Collection

Collections are active only during the time that they are referenced. Typically, a collection is referenced when a variable is set to the result of a command that creates it or when it is passed as an argument

to a command or a procedure.

For example, if you save a collection of ports, as shown in the following command:

```
nt_shell> set ports [get_ports *]
```

then either of the following two commands deletes the collection referenced by the *ports* variable:

```
nt_shell> unset ports
nt_shell> set ports "value"
```

Collections are implicitly deleted when they go out of scope. Collections go out of scope when the parent (or other antecedent) of the objects within the collection is deleted. If you query a collection that has been deleted, an error message similar to the following appears:

```
nt_shell> query_objects $ports
Error: No such collection '_sel26' (SEL-001)
```

Iteration

To iterate over the objects in a collection, use the **foreach_in_collection** command. You cannot use the Tcl-supplied **foreach** command to iterate over the objects in a collection, because the **foreach** command requires a list, and a collection is not a list. If you use the **foreach** command on a collection, it destroys the collection.

The arguments for the **foreach_in_collection** command are as follows: an iterator variable, the collections over which to iterate, and the set of commands to apply during each iteration. Unlike the **foreach** command, the **foreach_in_collection** command does not accept a list of iterator variables.

The following example is an iterative way to perform a query. For details, see the **foreach_in_collection** man page.

```
nt_shell> \
foreach_in_collection s1 $collection {
  echo [get_object_name $s1]
  }
```

Manipulating Collections

NanoTime provides commands to work with collections:

- **add_to_collection** - Adds objects or collections to a base collection.
- **append_to_collection** - Adds objects or collections to a base collection using a command syntax suitable for use inside loops.
- **compare_collections** - Verifies that two collections contain the same objects (optionally, in the same order).
- **copy_collection** - Creates a new collection containing the same objects (in the same order) as a given collection. Not all collections can be copied.
- **index_collection** - Extracts a single object from a collection and creates a new collection containing that object. Not all collections can be indexed.
- **remove_from_collection** - Takes a base collection and a list of element names or collections that you want to remove from it.

- **sizeof_collection** - Returns the number of objects in a collection.
- **sort_collection** - Reorders items in a collection based on attributes.

Filtering

You can filter any collection by using the **filter_collection** command. It takes a base collection and creates a new collection that includes only those objects that match an expression.

Many of the commands that create collections support a **-filter** option, which allows objects to be filtered out before they are ever included in the collection. Frequently this is more efficient than filtering after they are included in the collection.

The following example filters out all leaf cells:

```
nt_shell> filter_collection [[get_cells *] "is_hierarchical == true"]
{"i1", "i2"}
```

The basic form of a filter expression is a series of relations joined together with AND and OR operators. Parentheses are supported. The basic relation contrasts an attribute name with a value through a relational operator. In the previous example, **is_hierarchical** is the attribute, **==** is the relational operator, and **true** is the value.

The relational operators are as follows:

```
==   Equal
!=   Not equal
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
=~   Matches pattern
!~   Does not match pattern
```

The relational rules are as follows:

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only using the == and != operators. The value can be only **true** or **false**.

In addition, existence relations determine whether an attribute is defined for an object. For example,

```
sense == setup_clk_rise and defined(sdf_cond)
```

The existence operators are *defined* and *undefined*. These operators apply to any attribute that is valid for the object class.

Sorting Collections

You can sort a collection by using the **sort_collection** command. The command takes a base collection and a list of attributes as sort keys. The result is a copy of the base collection sorted by the given keys. Sort order is ascending by default, or descending when you specify the **-descending** option. The following example sorts the ports by direction and then by full name.

```
nt_shell> sort_collection [get_ports *] \
{direction full_name}
```

```
{"in1", "in2", "out1", "out2"}
```

Implicit Query of Collections

A command that creates a collection, when used at the beginning of a command line, implicitly queries the collection and displays a list of the collection contents. The number of objects displayed is controlled by the **collection_result_display_limit** variable.

```
nt_shell> set_input_delay 3.0 [get_ports in*]
1
nt_shell> get_ports in*
{"in0", "in1", "in2"}
nt_shell> query_objects -verbose [get_ports in*]
{"in0", "in1", "in2"}
nt_shell> set iports [get_ports in*]
{"in0", "in1", "in2"}
```

In the first example, the **get_ports** command creates a collection of ports that is passed to the **set_input_delay** command. This collection is not the result of the primary command (**set_input_delay**), so it is not queried. The second example shows how a command that creates a collection automatically queries the collection when that command is used as a primary command. The third example shows the verbose feature of the **query_objects** command, which is not available with implicit query. Finally, the fourth example sets the variable iports to the result of the **get_ports** command. Only in the final example does the collection persist to future commands. The collection persists until iports is overwritten, unset, or goes out of scope.

SEE ALSO

```
add_to_collection(2)
append_to_collection(2)
compare_collections(2)
copy_collection(2)
filter_collection(2)
foreach_in_collection(2)
index_collection(2)
query_objects(2)
remove_from_collection(2)
sizeof_collection(2)
sort_collection(2)
collection_result_display_limit(3)
```

compare_collections

Compares the contents of two collections.

SYNTAX

```
status compare_collections
   [-order_dependent]
   coll_A
   coll_B
```

Data Types

```
   coll_A         collection
   coll_B         collection
```

ARGUMENTS

-order_dependent

Considers the order of the objects during the comparison; the collections are considered to be different if the objects are ordered differently.

coll_A

Specifies the base collection for the comparison. An empty string (the empty collection) is allowed.

coll_B

Specifies the collection to compare to *coll_A*. An empty string (the empty collection) is allowed.

DESCRIPTION

The **compare_collections** command compares the contents of two collections. By default, the order of the objects does not matter, so that the collection of cells {u1 u2} is the same as the collection of cells {u2 u1}. The **-order_dependent** option causes the order of the objects to be considered.

Either or both of the collections can be the empty string (the empty collection). If two empty collections are compared, the command succeeds (that is, the **compare_collections** command considers the two collections to be identical), and NanoTime displays a status of 0.

COMMAND PHASING

The **compare_collections** command can be executed at any time.

EXAMPLES

The following example shows a variety of comparisons. A result of 0 from the **compare_collections** command indicates success. Any other result indicates failure.

```
nt_shell> compare_collections [get_cells *] [get_cells *]
0
nt_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
nt_shell> set c2 [get_cells {u2 u1}]
{"u2", "u1"}
nt_shell> set c3 [get_cells {u2 u4 u6}]
{"u2", "u4", "u6"}
nt_shell> compare_collections $c1 $c2
0
nt_shell> compare_collections $c1 $c2 -order_dependent
-1
nt_shell> compare_collections $c1 $c3
-1
```

The following example shows how empty collections are compared.

```
nt_shell> set c4 ""
nt_shell> compare_collections $c1 $c4
-1
nt_shell> compare_collections $c4 $c4
0
```

SEE ALSO

collections(2)

complete_net_parasitics

Completes partial parasitics annotated on all nets of the current design.

SYNTAX

```
status complete_net_parasitics
   [-complete_with completion_type]
```

Data Types

```
   completion_type         string
```

ARGUMENTS

-complete_with *completion_type*

Indicates that a net with partially annotated parasitics and missing segments is to be completed by inserting capacitances and resistances according to *completion_type*. Allowed values are **zero** (the default) which completes the net by inserting very small, near-zero values; and **rc**, which completes the net by inserting fixed capacitances and resistances specified by the **parasitics_completion_capacitance** and **parasitics_completion_resistance** variables. This option is equivalent to the **read_parasitics -complete_with** or **check_design -complete_with** commands.

DESCRIPTION

The **complete_net_parasitics** command completes all nets in the design that have incomplete RC networks annotated from SPEF or SPF files.

The **complete_net_parasitics** and **read_parasitics -complete_with** commands complete a net only if all missing segments are between two pins, and only if the nets are partially annotated (nets are not affected if they are fully annotated or have no annotation at all). Also, the net must be hierarchical, so that if the parasitics for the block-level parts of a net are missing, those parasitics could exist in the top-level net. If any of these conditions are not met, you must correct the SPEF or DSPF file manually.

Use the **report_annotated_parasitics** command to view how the parasitics are completed.

After the completion of the nets, there is no direct way to remove only the completed segments. You can remove all parasitics read and annotated with the **read_parasitics** and **complete_net_parasitics** commands by using the **remove_annotated_parasitics** command. In this case, you must then reread the previously annotated parasitics by using the **read_parasitics** command. Alternatively, the **reset_design** command removes all attributes from the current design, including annotated parasitics.

COMMAND PHASING

The **complete_net_parasitics** command can only be executed after the **link_design** command and before the **check_design** command.

EXAMPLES

The following example completes the partially annotated parasitics from the file rc_file.spef by inserting zero capacitances and resistances.

```
nt_shell> read_parasitics rc_file.spef
nt_shell> complete_net_parasitics -complete_with zero
```

SEE ALSO

read_parasitics(2)
remove_annotated_parasitics(2)
report_annotated_parasitics(2)
reset_design(2)
parasitics_completion_capacitance(3)
parasitics_completion_resistance(3)
parasitics_allow_spf_net_override(3)

continue_trace

Runs an additional iteration of path tracing during signal integrity analysis.

SYNTAX

```
status continue_trace
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command runs an additional iteration of path tracing during signal integrity (crosstalk) analysis. It can be used only after one or more path-tracing iterations have been run, and only when crosstalk analysis is enabled. You must have a NanoTime Ultra license to perform signal integrity analysis.

Use this command to gain additional accuracy by using the arrival windows calculated in the previous path-tracing iteration.

COMMAND PHASING

The **continue_trace** command can only be executed in the "paths-traced" state (after the **trace_paths** command has been run).

EXAMPLES

The following example uses the **continue_trace** command to execute an additional path-tracing iteration

and shows typical output from the iteration.

```
nt_shell> continue_trace
Continuing a tpath trace in Signal Integrity mode
Running SI tracing analysis iteration: 3

Performing Max Clock Search

Updating Clock Arrivals with pulse properties

Finished updating Clock Arrivals with pulse properties

Performing Min Clock Search
 ...

Reporting SI convergence criteria
--------------------------------
  Total number of nets   : 2100
  Number of coupled nets : 6
  Effective nets         : 6

  current iteration      : 3
  iteration limit        : 1

  Reevaluated  nets      : 4
  reevaluated net limit  : 0
  reevaluated net pct : 19.048
  reeval net pct limit: 0.000
  coupled reevaluated net pct      : 66.667
  coupled reevaluated net pct limit: 0.000

  max delta delay      : 0.381
  max delta delay limit: 0.000
  min delta delay      : -0.258
  min delta delay limit: 0.000

Continued SI tracing with iteration 3
1
```

SEE ALSO

```
report_paths(2)
trace_paths(2)
si_enable_analysis(3)
```

copy_collection

Duplicates the contents of a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection copy_collection
   base_collection
```

Data Types

base_collection	collection

ARGUMENTS

base_collection

Specifies the collection to be copied. If the empty string is used for the argument, the command returns the empty string (a copy of the empty collection is the empty collection).

DESCRIPTION

The **copy_collection** command creates a duplicate of an existing collection.

Rather than copy a collection, it is more efficient, and almost always sufficient, to simply have more than one variable reference the same collection. For example, if you create a collection and save a reference to it in variable *c1*, assigning the value of *c1* to another variable *c2* creates a second reference to the same collection:

```
nt_shell> set c1 [get_cells "U1*"]
{"U1", "U10", "U11", "U12"}
nt_shell> set c2 $c1
{"U1", "U10", "U11", "U12"}
```

This has not copied the collection. There are now two references to the same collection. If you change the *c1* variable, *c2* continues to reference the original collection:

```
nt_shell> set c1 [get_cells "block1"]
{"block1"}
nt_shell> query_objects $c2
{"U1", "U10", "U11", "U12"}
```

There might be cases when you genuinely want a second collection. Use the **copy_collection** command to create a new collection that is a duplicate of the original.

COMMAND PHASING

The **copy_collection** command can be executed at any time.

EXAMPLES

The following example shows the result of copying a collection. Functionally, it is not much different from having multiple references to the same collection.

```
nt_shell> set c1 [get_cells "U1*"]
{"U1", "U10", "U11", "U12"}
nt_shell> set c2 [copy_collection $c1]
{"U1", "U10", "U11", "U12"}
nt_shell> unset c1
nt_shell> query_objects $c2
{"U1", "U10", "U11", "U12"}
```

SEE ALSO

collections(2)

cputime

Retrieves the overall user time associated with the current nt_shell process.

SYNTAX

```
status cputime
   [-components]
   [-include_elapsed_time]
```

ARGUMENTS

-components

Reports the user and system components of CPU time.

-include_elapsed_time

Reports CPU time and elapsed time.

DESCRIPTION

The **cputime** command returns the accumulated user time, in seconds, for the current nt_shell process. An integer number of seconds is returned.

EXAMPLES

```
nt_shell> cputime
374
```

SEE ALSO

mem(2)

create_clock

Creates a clock object and assigns it to a port or boundary pin in the design.

SYNTAX

```
status create_clock
   -period period_value
   [-name clock_name]
   [-rise time]
   [-fall time]
   [-waveform edge_list]
   [source_object]
```

Data Types

```
   period_value     float
   clock_name       string
   time             float
   edge_list        list
   source_object    list
```

ARGUMENTS

-period *period_value*

Specifies the clock period in time units of the technology. This is the time period over which the clock waveform repeats.

-name *clock_name*

Specifies the name of the clock being created. If you do not use this option, the clock gets the same name as the first clock source object specified in the command. If you do not specify source objects, you must use the **-name** option, which creates a virtual clock not associated with any port or pin. You can use the **-name** option along with a source object to give the clock a name that is different from the source pin or port.

-rise *time*

Specifies the time at which the rising edge occurs. It must be greater than or equal to 0.0 and less than or equal to the clock period.

-fall *time*

Specifies the time at which the falling edge occurs. It must be greater than or equal to 0.0 and less than or equal to the clock period.

-waveform ***edge_list***

Specifies the rising and falling edge times of the clock waveform, in time units of the technology, within one clock period. This is an alternative syntax for specifying the **-rise** and **-fall** options that is compatible with the Synopsys Design Constraints (SDC) format. You must specify exactly two edge times. The first time specifies the rising edge and the second time specifies the falling edge. The falling edge time can be earlier than the rising edge time. If you do not use the **-waveform** option or **-rise** and **-fall** options, the rising edge defaults to time 0.0 and the falling edge defaults to a time equal to one-half the clock period.

source_object

Specifies the clock definition point in the design: a port or transistor gate pin where the clock exists. Only boundary pins can be used. Each port and transistor gate pin in the design should have no more than one clock defined on it. If you do not specify any source object, NanoTime creates a virtual clock not associated with any port or transistor gate pin.

DESCRIPTION

The **create_clock** command creates a clock object and applies that clock to a specified source object (a port or boundary pin in the design). NanoTime traces the clock network so that the clock reaches all sequential elements in the transitive fanout of the source. If you want to define a clock on an internal pin, use the **create_generated_clock** command instead.

If you do not specify a clock source object, the command creates a virtual clock that can be used to represent an off-chip clock for specifying input delays and output delays. For more information, see the man pages for the **set_input_delay** and **set_output_delay** commands.

You must specify the period of the clock using the **-period** option. By default, the clock has a single pulse per period and a duty cycle of 50 percent, with a rising edge at the start and end of the clock period and a falling edge at the middle. To specify an asymmetric or inverted waveform, use the **-rise** and **-fall** options or the **-waveform** option. NOTE: The **-rise** and **-fall** options might not be supported by other tools that use Synopsys Design Constraints (SDC) commands.

By default, NanoTime calculates clock latency by propagating delays along the clock network. To use ideal clocking instead, set the **timing_all_clocks_propagated** variable to **false** before you create the clocks. To use ideal clocking only on specified clocks, ports, or pins, use the **remove_propagated_clock** command on those objects. To specify the latency and transition times for ideal clocks, use the **set_clock_latency** and **set_clock_transition** commands.

The specified clock source is propagated after the **match_topology** command. By default, NanoTime does not check logic values set by logic constraints or by the **set_case_analysis** command. To allow NanoTime to consider logic values during clock propagation, set the **topo_clock_propagation_strict_logic_check** variable to **true** before executing the **match_topology** command.

To show information about clocks in the design, use the **report_clock** command. To create a collection of clocks matching a pattern and optional filter criteria, use the **get_clocks** command.

To undo the effects of the **create_clock** command, use the **remove_clock** command.

NOTE: The **create_clock** command is a Synopsys Design Constraints (SDC) command. However, the **-rise** and **-fall** options might not be supported by other tools that use SDC commands. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **create_clock** command is typically executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command). It cannot be used before the **link_design** command. If you execute the **create_clock** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If you modify the clock network after the first **match_topology** command, you might need to force another clock propagation operation before trying to mark or recognize topologies. To do this, set the **topo_match_topology_reset_clock_and_topology** variable to **true** and execute the **match_topology -force** command to cause NanoTime to repropagate the clock network and rerecognize topologies.

If you use the **create_clock** command in a topology object created with the **create_lib_topology** command, specify the **-action_pre_match_topology** option.

EXAMPLES

The following command creates a clock on port PHI1 with a period of 10.0, a rising edge at time 5.0, and a falling edge at time 9.5. The clock name is the same as the port name.

```
nt_shell> create_clock PHI1 -period 10.0 -rise 5.0 -fall 9.5
```

The following command achieves the same result as the previous one. It uses the **-waveform** option instead of the **-rise** and **-fall** options to define the same clock.

```
nt_shell> create_clock PHI1 -period 10.0 -waveform {5.0 9.5}
```

The following command creates a clock named PHI2 on input port ck2, with a period of 10.0, a rising edge at time 6.0, and a falling edge at time 10.0. The clock name is different from the port name.

```
nt_shell> create_clock -name PHI2 -period 10.0 -waveform {6.0 10.0} \
          [get_ports ck2]
```

The following command creates a virtual clock named PHI2 with a period of 10.0, a rising edge at time 0.0, and a falling edge at time 5.0. A virtual clock is not associated with any port or pin.

```
nt_shell> create_clock -name PHI2 -period 10.0 -waveform {0.0 5.0}
```

The following example creates a clock named clk2 at multiple sources in the design, with a period of 10.0, a rising edge at time 2.0, and a falling edge at time 4.0.

```
nt_shell> create_clock -name clk2 -period 10.0 -waveform {2.0 4.0} \
          [get_pins {clkgen1.mp.gate  clkgen2.mn.gate}]
```

SEE ALSO

```
all_clocks(2)
get_clocks(2)
create_generated_clock(2)
remove_clock(2)
remove_propagated_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
set_input_delay(2)
set_output_delay(2)
set_propagated_clock(2)
topo_match_topology_reset_clock_and_topology(3)
topo_clock_propagation_strict_logic_check(3)
```

create_command_group

Creates a new command group.

SYNTAX

```
string create_command_group [-info info_text]
group_name
```

ARGUMENTS

-info *info_text*

Help string for the group

group_name

Specifies the name of the new group.

DESCRIPTION

The **create_command_group** command is used to create a new command group, which you can use to separate related user-defined procedures into functional units for the online help facility. When a procedure is created, it is placed in the "Procedures" command group. With the **define_proc_attributes** command, you can move the procedure into the group you created.

The *group_name* can contain any characters, including spaces, as long as it is appropriately quoted. If *group_name* already exists, **create_command_group** quietly ignores the command. The result of **create_command_group** is always an empty string.

EXAMPLES

The following example demonstrates the use of the **create_command_group** command:

```
prompt> create_command_group {My Procedures} -info "Useful utilities"

prompt> proc plus {a b} { return [expr $a + $b] }

prompt> define_proc_attributes plus -command_group "My Procedures"

prompt> help
My Procedures:
  plus

 ...
```

SEE ALSO

define_proc_attributes(2)
help(2)
proc(2)

create_gated_clock_timing_check

Creates a gated clock timing check.

SYNTAX

```
status create_gated_clock_timing_check
   [-label name]
   -from from_object
      | -rise_from from_object
      | -fall_from from_object
   -to to_object
      | -rise_to to_object
      | -fall_to to_object
   [-setup | -hold]
   [-use_existing_timing_points]
   check_value
```

Data Types

```
   name              string
   from_object       list
   to_object         list
   check_value       float
```

ARGUMENTS

-label *name*

An optional label used to identify the timing check.

-from *from_object*

Specifies a pin or port in the current design as the clock signal being gated. You must specify one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising transitions at the clock pin.

-fall_from *from_object*

Similar to the **-from** option, but applies only to falling transitions at the clock pin.

-to ***to_object***

Specifies a pin or port in the current design as the clock-gating signal. You must specify one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***to_object***

Similar to the **-to** option, but applies only to rising signals at the clock-gating pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling signals at the clock-gating pin.

-setup

Indicates that the data check value is for setup analysis only.

-hold

Indicates that the data check value is for hold analysis only. If neither of the **-setup** or **-hold** options is specified, the value applies to both setup and hold.

-use_existing_timing_points

When this option is used, the timing check creation must only specify pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is true for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **create_gated_clock_timing_check** command has been issued.

check_value

Specifies the value of the setup or hold time for the check.

DESCRIPTION

NanoTime performs clock gating setup and hold checks at clock gates that it recognizes. If there is a clock gate that NanoTime does not recognize, you can manually specify a clock gating check for that clock gating structure with the **create_gated_clock_timing_check** command.

The command specifies a "from" object, a "to" object, and the setup or hold time. The "from" object is a port or pin where there is a clock signal. The "to" object is a port or pin where there is a clock gating signal. NanoTime performs the timing checks at the specified "from" and "to" points.

For a setup check, NanoTime verifies that the clock gating signal changes at least a specified amount of time before a clock transition occurs. This ensures that a glitch will not occur at the gated clock signals.

For a hold check, NanoTime verifies that the clock gating signal remains stable for at least the specified amount of time after a clock transition occurs. This ensures that the active clock signal is not clipped.

To modify a gated clock timing check set with the **create_gated_clock_timing_check** command, use

the **set_gated_clock_timing_check_attributes** command. To remove a gated clock timing check, use the **remove_gated_clock_check** command.

A timing check specified by this command is a non-transparent timing constraint. It is different than an edge-triggered timing check at a flip-flop specified by the **create_timing_check** command. Though no warning message is issued by NanoTime, intermingling edge-triggered and non-transparent timing checks at a timing point can lead to unexpected results.

COMMAND PHASING

In a typical NanoTime flow,the **create_gated_clock_timing_check** command must be executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design** commands).

If you use the command after the **check_design** command, NanoTime ordinarily transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command. However, if you use the **-use_existing_timing_points** option, you can use the **create_gated_clock_timing_check** command after the **check_design** command has been issued.

EXAMPLES

Assume a design with two input ports, CLK (a rising-edge clock signal) and CLKGATE (a clock enable signal). The two signals feed into an AND gate. The clock signal is enabled when CLKGATE is high. The clock gating signal must be stable (either high or low) at least 0.5 time units before a rising transition at CLK, and must remain stable for at least 0.7 time units after the rising transition at CLK. To specify these checks, use the following commands:

```
nt_shell> create_gated_clock_timing_check \
          -rise_from [get_ports CLK] \
          -to [get_ports CLKGATE] -setup 0.5

nt_shell> create_gated_clock_timing_check \
          -rise_from [get_ports CLK] \
          -to [get_ports CLKGATE] -hold 0.7
```

SEE ALSO

```
remove_gated_clock_check(2)
report_paths(2)
set_gated_clock_timing_check_attributes(2)
trace_paths(2)
create_timing_check(2)
timing_clock_gate_hold_fall_margin(3)
```

create_generated_clock

Creates a generated clock object and defines its timing relationship to a master clock.

SYNTAX

```
status create_generated_clock
   [-name clock_name]
   [-source master_pin]
   [-divide_by divide_factor |
      -multiply_by multiply_factor |
      -edges edge_list]
   [-edge_shift edge_shift_list]
   [-duty_cycle percent]
   [-invert]
   [-master_clock clock]
   source_objects
```

Data Types

clock_name	string
master_pin	list
divide_factor	integer
multiply_factor	integer
edge_list	list
edge_shift_list	list
percent	float
clock	string
source_objects	list

ARGUMENTS

-name *clock_name*

Specifies the name of the generated clock. If you do not use this option, the clock receives the same name as the first clock source object specified in the command.

-source *master_pin*

Specifies the master clock source (a clock source pin in the design) from which the clock waveform is derived. NanoTime calculates the latency of the generated clock by considering the latency of the *master_pin*, and the delay from that pin to the generated clock's *source_objects*. Two source pins can be specified when needed for differential generated clocks. These pins must be on nets that are already

declared as a differential net pair.

-divide_by *divide_factor*

Specifies the frequency division factor. If the *divide_factor* value is 2, the generated clock frequency is half that of the master clock.

-multiply_by *multiply_factor*

Specifies the frequency multiplication factor. If the *multiply_factor* value is 2, the generated clock frequency is twice that of the master clock.

-edges *edge_list*

Specifies the generated waveform as a list of three integers. These integers represent the edges from the master clock that are used to trigger the edges of the generated clock, where 1 represents the first master edge (the first rising edge), 2 represents the second master edge (the first falling edge), and 3 represents the third master edge (the second rising edge). The three specified edges trigger the first rising, first falling, and second rising edges of the generated clock, respectively. When the list contains a repeated digit (for example, {1 1 3}), it generates a pulse clock; the leading and trailing edges of the pulse are triggered by the same master edge.

-edge_shift *edge_shift_list*

A list of floating-point values that shift the edges specified by the **-edges** option later in time by the amount specified, in time units, to generate the clock edges of the generated clock. For example, **-edges {1 4 5} -edge_shift {1.1 1.3 1.4}** specifies edges 1, 4 and 5 of the source clock, shifted to the right by 1.1, 1.3, and 1.4 time units, respectively, to generate the rising, falling, and rising edges of the generated clock. This option cannot be used with a pulse clock.

-duty_cycle *percent*

Specifies the duty cycle, in percent, if the **-multiply_by** option is used. The duty cycle is the high pulse width divided by the period, multiplied by 100.

-invert

Inverts the clock signal generated by the **-divide_by** or **-multiply_by** option.

-master_clock *clock*

When a master clock is specified without any -source pin, the master clock is assumed to be a virtual clock and the generated clock is a transformation of this virtual clock. A master clock can also be used for this generated clock if a -source pin is specified. This will allow the user to determine which incoming clock to use if multiple clocks fan into the source object of the generated clock. It also allows the period attribute to be available after check_topology.

source_objects

Specifies the clock definition point in the design: a port, pin, or net where the generated clock exists (typically, a single pin). If you specify a net, NanoTime considers the driver pin of the net to be the generated clock source.

DESCRIPTION

The **create_generated_clock** command creates a clock object and applies that clock to a specified source object in the current design. The timing characteristics of the generated clock (period, waveform, and latency) depend on another clock in the design, called the master clock. If you change the characteristics of the master clock, the generated clock characteristics also change accordingly.

For example, if the design has a main clock SYSCLK and a divide-by-2 clock generator, you can define the divide-by-2 generated clock as follows:

```
nt_shell> create_generated_clock -name CLK2 \
          -source [get_ports SYSCLK] \
          -divide_by 2 [get_pins FF1/Q]
```

In this example, SYSCLK is the master clock port and the master source of the generated clock. FF1/Q is the output pin where the divide-by-2 clock is generated and is the generated clock source object. The period of the generated clock is twice that of the master clock, as specified by the **-divide_by** option.

You specify two source objects in the design: the generated source object (where the generated clock is defined) and the master source object (where the master clock is defined). The generated source object can be specified as an input port, pin, or net. If it is specified as a net, the driving pin of the net is considered the clock source object. If the object already has a generated clock, the older one is overwritten by the new one.

The master clock source object is usually specified with the **-source** option. However, if the **-source** option is not used, then the master clock must be a virtual clock (a clock created by a **create_clock** command without specifying a source object). In that case, in the **create_generated_clock** command, specify the name of the virtual clock using the **-master_clock** option.

You specify the timing relationship between the master clock and generated clock by using one of the following options:

- **-divide_by** (frequency divider)
- **-multiply_by** (frequency multiplier)
- **-edges** (edge-derived generated clock)

The **-divide_by** option specifies the generated clock by dividing the frequency of the master clock by an integer (for example, a divide-by-2 clock divider). If the specified divide-by factor is a power of 2 (2, 4, 8, 16, ...), NanoTime uses just the rising edges of the master clock to determine the edges of the generated clock. For all other factors, NanoTime scales all the edge times of the master clock to determine the edge times of the generated clock.

The **-multiply_by** option specifies the generated clock by multiplying the frequency of the master clock by an integer (for example, a multiply-by-2 phase-locked loop clock generator).

A frequency-divided or frequency-multiplied clock can be inverted by using the **-invert** option.

The **-edges** option lists three edge numbers of the master clock that are used to trigger the edges of the generated clock. For example, specifying the **-edges {1 5 6}** option means that the first rising, first falling, and second rising edges of the generated clock are triggered by master clock edges as follows: the

first edge (the first edge is always a rising edge), the fifth overall edge (which is the third rising edge), and the sixth overall edge (which is the third falling edge).

A repeated digit in the list generates a pulse clock. In that case, the specified edge of the master clock triggers both the leading and trailing edges of the generated clock pulse. For example, **-edges {1 1 3}** means that the rising and falling edges of the initial pulse of the generated clock are both triggered by the first edge of the master clock, and the next rising edge of the generated clock is triggered by the third edge of the master clock. Nominally this represents a pulse width of zero. However, the clock generator circuit should be designed with different delays so that the leading edge of the pulse occurs before the trailing edge.

Specifying the generated clock as a pulse clock (rather than simply specifying the generated clock edge times) ensures correct checking of delays between the domains of the master clock and generated clock.

The position (even or odd) of the repeated digit in the list determines whether an active-high or active-low pulse is generated, and the edge number (even or odd) determines the type of edge in the master clock used to trigger the pulse. For example:

- `-edges {1 1 3}`
  `rising edge of master triggers active-high pulse`

- `-edges {2 2 4}`
  `falling edge of master triggers active-high pulse`

- `-edges {1 3 3}`
  `rising edge of master triggers active-low pulse`

- `-edges {2 4 4}`
  `falling edge of master triggers active-low pulse`

The rise and fall latencies of a pulse clock must be defined to produce a positive pulse width. For example, for a rising pulse:

```
nt_shell> create_generated_clock -name CLKP \
          -source SCLK -edges {1 1 3} \
          [get_pins U1/Z]
nt_shell> set_clock_latency -source -rise 0.0 \
          [get_clocks CLKP]
nt_shell> set_clock_latency -source -fall 0.2 \
          [get_clocks CLKP]
```

If you are using the **-edges** option to generate a normal-edge clock (not a pulse clock), you can shift each edge of the generated clock by a specified amount of time using the **-edge_shift** option. You should use the **-edge_shift** option only to represent the intentional operation of the clock generator circuit, not to represent clock latency.

The following commands can reference a generated clock: **set_clock_latency**, **set_clock_uncertainty**, **set_propagated_clock**, and **set_clock_transition**.

For internally generated clocks, NanoTime automatically computes the clock source latency if the master clock of the generated clock has propagated latency and no user-specified value for generated clock source latency exists. If the master clock is ideal and has source latency, and there is no user-specified value for the generated clock's source latency, then zero source latency is assumed.

To display information about generated clocks, use the **report_clock** command. To undo the effects of the **create_generated_clock** command, use the **remove_clock** command.

NOTE: The **create_generated_clock** command is a Synopsys Design Constraints (SDC) command. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **create_generated_clock** command is typically executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command). It cannot be used before the **link_design** command. If you execute the **create_generated_clock** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If you modify the clock network after the first **match_topology** command, you might need to force another clock propagation operation before trying to mark or recognize topologies. To do this, set the **topo_match_topology_reset_clock_and_topology** variable to **true** and execute the **match_topology -force** command to cause NanoTime to repropagate the clock network and rerecognize topologies.

If you use the **create_generated_clock** command in a topology object created with the **create_lib_topology** command, specify the **-action_pre_match_topology** option.

EXAMPLES

The following command creates a virtual generated clock. Specifying the source pin is optional if the master clock is a virtual clock. (The generated clock uses only the user-defined clock latency of the master clock.)

```
nt_shell> create_generated_clock \
          -master_clock ref_clk \
          -edges {1 2 3} [get_port clk]
```

The following command creates a generated pulse clock with an active-high pulse. The rising edge of the master clock triggers the rising and falling edges of the pulse.

```
nt_shell> create_generated_clock -name CG1 \
          -master_clock ref_clk -edges {1 1 3} \
          [get_ports pclk]
nt_shell> set_clock_latency -source -rise 0.0 \
          [get_clocks CG1]
nt_shell> set_clock_latency -source -fall 0.2 \
          [get_clocks CG1]
```

The following command creates a generated clock with an active-high pulse. The falling edge of the master clock triggers the rising and falling edges of the pulse.

```
nt_shell> create_generated_clock \
          -master_clock ref_clk -edges {2 2 4} \
          [get_ports pclk]
```

The following command creates a generated clock with an active-low pulse. The falling edge of the master clock triggers the falling and rising edges of the pulse.

```
nt_shell> create_generated_clock \
          -master_clock ref_clk -edges {2 4 4} \
          [get_port pclk_n]
```

The following command creates a generated clock whose edges occur at edges 1, 3, and 5 of the master clock source.

```
nt_shell> create_generated_clock -edges {1 3 5} \
          -source [get_pins CLK] [get_pins signal_A]
```

The following command creates the generated clock in the previous example, but with each derived edge shifted by 1.0 time unit.

```
nt_shell> create_generated_clock -edges {1 3 5} \
          -edge_shift {1.0 1.0 1.0} \
          -source [get_pins CLK] [get_pins signal_A]
```

The following command creates an inverted clock.

```
nt_shell> create_generated_clock \
          -divide_by 1 -invert \
          -source [get_pins CLK] \
          [get_pins CLKINV]
```

The following command creates a frequency divide-by-2 generated clock.

```
nt_shell> create_generated_clock \
          -divide_by 2 \
          -source [get_pins CLK] \
          [get_pins div2/Q]
```

The following command creates a frequency divide-by-3 generated_clock. If the master clock period is 30 and master waveform is {24 36}, the generated clock period is 90 with waveform {72 108}.

```
nt_shell> create_generated_clock \
          -divide_by 3 \
          -source [get_pins CLK] \
          [get_pins div3/Q]
```

The following command creates a frequency multiply-by-2 generated clock with a duty cycle of 60 per cent.

```
nt_shell> create_generated_clock \
          -multiply_by 2 -duty_cycle 60 \
          -source [get_pins CLK] \
          [get_pins signal_B]
```

The following example creates a frequency multiply-by-3 generated clock with a duty cycle equal to the master clock duty cycle. If the master clock period is 30 and master waveform is {24 36}, the generated clock period is 10 with waveform {8 12}.

```
nt_shell> create_generated_clock \
          -multiply_by 3 \
          -source [get_pins CLK] \
          [get_pins div3/Q]
```

SEE ALSO

```
check_topology(2)
check_design(2)
create_clock(2)
create_lib_topology(2)
get_generated_clocks(2)
remove_generated_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
set_propagated_clock(2)
topo_match_topology_reset_clock_and_topology(3)
```

create_lib_topology

Creates a new lib_topology object in a topology library and specifies the action taken when a match occurs. Use this command only in a .libt lib_topology file.

SYNTAX

```
status create_lib_topology
   [-version version_string]
   [-description description_string]
   [-enable_search]
   [-subckt_name name_list]
   [-current_design]
   [-search_priority priority]
   [-action_pre_match_topology command]
   [-action_post_clock_propagation command]
   [-action_post_match_topology command]
   [-action_pre_check_topology command]
   [-action_post_check_topology command]
   [-action_pre_check_design command]
   [-action_post_check_design command]
```

Data Types

```
   version_string            string
   description_string        string
   name_list                 string
   priority                    string
   command                   string
```

ARGUMENTS

-version *version_string*

A user-defined version string for the new lib_topology object. This string is for documentation only and does not affect the analysis results.

-description *description_string*

A brief user-defined description of the new lib_topology object. This string is for documentation only and does not affect the analysis results.

-enable_search

Sets the lib_topology to be initially enabled for searching when the library is loaded with the **read_topology_library** command. Otherwise, searching for the topology is disabled by default and can be enabled by the **set_search_enabled** command.

-subckt_name ***name_list***

A list of subcircuit names used for subcircuit name matching. Wildcard characters are allowed. NanoTime finds the named SPICE subcircuit in the design netlist. If neither this option nor **-current_design** option is used, you must provide a search pattern in a .sp file.

-current_design

Sets NanoTime to apply action commands on the current design. If this option is not used, you must provide subcircuit names for subcircuit name matching or a search pattern in a .sp file.

-search_priority ***priority***

Assigns a priority string to the lib_topology object. NanoTime searches for topologies in alphanumeric order based on the search priority string of the lib_topology, if specified, or by the lib_topology name otherwise. NanoTime always searches for topologies that have a search priority string before those that do not have one.

-action_pre_match_topology ***command***

The command is executed before the **match_topology** phase point, typically to perform clock tree propagation adjustments needed for correct marking of clocked topologies.

-action_post_clock_propagation ***command***

The command is executed after the clock propagation phase point, typically to perform topology marking.

-action_post_match_topology ***command***

The command is executed after the **match_topology** phase point, typically to perform topology marking.

-action_pre_check_topology ***command***

The command is executed before the **check_topology** phase point, typically to perform adjustments needed for correct marking of clocked topologies.

-action_post_check_topology ***command***

The command is executed after the **check_topology** phase point, typically to perform topology marking.

-action_pre_check_design ***command***

The command is executed before the **check_design** phase point, typically to adjust topology settings of marked topologies.

-action_post_check_design ***command***

The command is executed after the **check_design** phase point, typically to adjust timing constraints generated from marked topologies.

DESCRIPTION

The **create_lib_topology** command creates a new lib_topology object. The command should be used only in the .libt file in a topology library directory, not at the nt_shell prompt. The command and its associated files specify an application mode (either current design mode or structure searching mode), a search method (either subcircuit matching or pattern matching), a structure to be found in the design, and a set of actions to be performed on current design or each instance of the structure found. During execution of **match_topology** or **check_topology**, NanoTime uses this information to identify and mark structures in the design.

Application Mode

There are two modes in which the lib_topology can be used: current design mode and searching mode. In the current design mode, the action commands are applied to the current design. In the searching mode, NanoTime searches the structures using the specified search method. The action commands are applied to the identified structures.

Search Method

There are two methods for searching and identifying structures in the design: subcircuit matching and pattern matching. The **foreach_match** command also uses these two searching methods.

In the subcircuit matching method, NanoTime searches for SPICE subcircuits that match the name provided with the **-subckt_name** option. The name can be specified with wildcard characters, for example, "lat_s*".

In the pattern matching method, NanoTime searches for a transistor structure in the netlist that matches a given SPICE pattern file provided in the topology library directory. The pattern file must be present in the topology library directory when the library is read in with the **read_topology_library** command.

The pattern file must have the same base name as the lib_topology file, with the extension .sp rather than .libt. The file must contain one or more subcircuit definitions using SPICE netlist syntax. The subcircuit names of the search patterns apply only to the local lib_topology definition.

Multiple pattern definitions can be used to represent variations of the basic netlist structure of the lib_topology object. NanoTime searches for each type of pattern in the order that it appears in the pattern file and applies the action commands to each successful match. The net and device names used in multiple subcircuit definitions must all be consistent with the action commands.

Action Scripts and Phase Points

The **create_lib_topology** command specifies a list of NanoTime commands that are to be executed at various phases of the analysis flow for current design or the identified structures. At least one action should be defined that marks one or more topology objects in the matching netlist pattern. NanoTime executes the commands within each match context in the same manner as the **foreach_match** command.

The commands can be executed at any of seven phase points, as specified in the following **create_lib_topology** command options:

-action_pre_match_topology: Used to perform clock tree propagation adjustments needed for the

correct marking of clocked topologies; has priority over automatic recognition.

-action_post_clock_propagation: Used to perform topology marking after clock propagation; can be used to remove the results of an automatic recognition.

-action_post_match_topology: Used to perform topology marking after built-in recognition. Automatic recognition takes precedence. Action script can remove any automatically-recognized topology.

-action_pre_check_topology: Used to perform clock tree propagation adjustments needed for the correct marking of clocked topologies. Library topologies take precedence over automatic recognition.

-action_post_check_topology: Used to perform topology marking after built-in recognition. Automatic recognition takes precedence. An action script can remove any automatically-recognized topology.

-action_pre_check_design: Used to adjust topology settings of marked topologies. Actions are taken before **check_design** activities are launched.

-action_post_check_design: Used to adjust timing constraints generated from marked topologies. The action scripts in this phase can verify that marked topologies generated the expected timing constraints.

NanoTime tracks all the matched patterns and executes the corresponding action commands at the designated phases of the analysis. For example, the command specified with the **-action_post_match_topology** option is executed only after the **match_topology** command. The "pre" phase commands are executed before any NanoTime built-in actions are taken, and the "post" phase commands are executed after any NanoTime built-in actions have been processed.

COMMAND PHASING

The **create_lib_topology** command is not phase-restricted.

EXAMPLES

The following command in a file muxflop.libt performs a **mark_latch** operation after clock propagation for each occurrence of the muxflop subcircuit.

```
create_lib_topology \
  -description "muxflop latch" \
  -version "V20070215" \
  -subckt_name muxflop \
  -enable_search \
  -action_post_clock_propagation { \
     mark_latch -latch_net lat1n -output lat1 \
     -feedforward { X5.Mn0 X5.Mp0 } \
     -feedback { Mp6 Mp7 Mn6 Mn7 } \
     -clock { Mn0.G Mp0.G } -name muxflop \
  }
```

SEE ALSO

```
check_topology(2)
create_topology_library(2)
list_topology_library(2)
match_topology(2)
read_topology_library(2)
set_search_enabled(2)
topo_library_order(3)
```

create_memory

Creates a memory object for timing analysis in NanoTime for memories.

SYNTAX

```
status create_memory
   -name memory_name
   -mode read | write | skip_array
   [-bitcell_ports ports]
   [cell]
```

Data Types

```
   memory_name          string
   ports                list
   cell                 string
```

ARGUMENTS

-name *memory_name*

Specifies a unique name for the memory being created.

-mode read | write | skip_array

Specifies the memory mode of operation to be analyzed: read, write, or skip_array. Delay calculation and timing checks are specific to this mode.

-bitcell_ports *ports*

Specifies the type of ports used by the bitcell within the memory array. This option takes a list of types describing one or more ports of each bitcell. Use a list of one type for a single port SRAM or a list of two types for a a dual port SRAM. The default is **nmos_bidi**.

cell

Specifies the top-level hierarchical cell to search to find the bitcells, wordlines and bitlines of the memory. If this argument is not present, the entire design is searched.

DESCRIPTION

The **create_memory** command in NanoTime for memories creates an object to enable memory analysis on the region of the design where the bitcells reside.

Bitcell identification is based on first finding ram topologies, then finding interconnections among the ram topologies in a pattern that is consistent with a memory. If a cell name appears in the argument list, the search for bitcells is limited to that hierarchical cell. The bitcell search also identifies the wordlines and bitlines of the memory core.

When the argument to the **-mode** option is **skip_array**, paths will not be traced through the bitcell array. There will be no write-mode HSPICE bit-column simulations, and only one read-mode simulation will exist. The read-mode simulation will measure the delay from the sense-amp enable to the sense-amp output only.

In the other modes of operations, the **-mode** option specifies whether path tracing and timing checks analyze the read or write operation. NanoTime analyzes only one mode at a time: read and write operations must be performed in separate passes. In addition, you must use the **set_case_analysis** command to put the memory into the corresponding mode of operation for the analysis.

You can optionally specify port types with the **-bitcell_ports** option. If the option is absent, the default is **nmos_bidi**. The **skip_array** mode analysis assumes 6T SRAM cells, so it supports only a single **nmos_bidi** port type. For more information about memory topologies, see the NanoTime for Memories User Guide. Valid values of the **-bitcell_ports** option are as follows:

```
nmos_read
nmos_write
nmos_bidi
two_nmos_read
single_ended_nmos_read
single_ended_nmos_read_differential_write
single_ended_two_nmos_read
```

COMMAND PHASING

The **create_memory** command must be executed before the **match_topology** command.

EXAMPLES

This command searches the entire design for bitcells, creates a memory called mem1, and specifies read mode analysis.

```
nt_shell> create_memory -name mem1 -mode read
```

SEE ALSO

```
remove_memory(2)
get_memory(2)
report_memory(2)
```

create_net_group

Creates a grouping of nets used for suppressing the reporting of similar paths.

SYNTAX

```
status create_net_group
   -name group_name
   net_list
```

Data Types

```
   group_name      string
   net_list        list
```

ARGUMENTS

-name *group_name*

A name assigned to the group of nets.

net_list

A list of nets in the design to be considered a group.

DESCRIPTION

The **create_net_group** command assigns a list of nets in the design to a group. You must specify a name for the group.

Each net can belong to no more than one group.

When you use the **report_paths** command with the **-suppress_similar_paths** option, NanoTime suppresses the reporting of multiple paths that have the same delay characteristics and have one or more nets belonging to the same group in the same position along the paths.

COMMAND PHASING

The **create_net_group** command can be executed any time after the **link_design** command.

EXAMPLES

The following command assigns all nets whose names start with the letters SB to a net group called "SBUS".

```
nt_shell> create_net_group -name SBUS {SB*}
```

SEE ALSO

```
list_net_groups(2)
remove_net_group(2)
```

create_port

Creates ports on specified nets at the top level of a flat netlist.

SYNTAX

```
status create_port
   [-input]
   [-output]
   [-inout]
   net_list
```

Data Types

```
   net_list        list
```

ARGUMENTS

-input

Creates an input port.

-output

Creates an output port.

-inout

Creates a bidirectional port.

net_list

The list of top-level design nets for which to make ports, one port per net.

DESCRIPTION

This command creates a port at each of the listed nets. You must specify **-input**, **-output**, or **-inout** to

create input ports, output ports, or bidirectional ports. The name of each new port matches the net name.

It is necessary to create ports when the design is a lower-level SPICE netlist that has no top-level subcircuit (in other words, a flat netlist). Because there are no ports, you must create them with the **create_port** command.

COMMAND PHASING

The **create_port** command can only be executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command). If the command is used at a later point, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations executed after the **check_topology** command.

EXAMPLES

The following example creates new inputs port connected to nets Sn1, Sn2, and Sn3.

```
nt_shell> create_port -input {Sn0 Sn1 Sn2}
1
nt_shell> report_port
 ...
               Min Rise Min Fall Max Rise Max Fall
Port    Dir    Cap      Cap      Cap      Cap
-------------------------------------------------
...
Sn0     in      --       --       --       --
Sn1     in      --       --       --       --
Sn2     in      --       --       --       --
Sn3     in      --       --       --       --
 ...
```

SEE ALSO

```
get_ports(2)
report_port(2)
```

create_timing_check

Creates a setup or hold timing check from a clock port or clock pin to a data port or data pin.

SYNTAX

```
status create_timing_check
   [-label name]
    -from from_object
      | -rise_from from_object
      | -fall_from from_object
    -to to_object
      | -rise_to to_object
      | -fall_to to_object
   [-trigger trigger_object
      | -rise_trigger trigger_object
      | -fall_trigger trigger_object]
   [-setup | -hold]
   [-continue]
   [-use_existing_timing_points]
   check_value
```

Data Types

```
name                string
from_object         list
to_object           list
trigger_object      list
check_value         float
```

ARGUMENTS

-label *name*

An optional label used to identify the timing check.

-from *from_object*

Specifies the clock port or clock pin in the current design for the timing check. You must specify one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising transitions at the clock pin.

-fall_from ***from_object***

Similar to the **-from** option, but applies only to falling transitions at the clock pin.

-to ***to_object***

Specifies the data pin or port in the current design for the timing check. You must specify one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***to_object***

Similar to the **-to** option, but applies only to rising signals at the data pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling signals at the data pin.

-trigger ***trigger_object***

The trigger port or pin at the input of the simulation unit related to the timing check, where path tracing starts. This option can only be used if the **-use_existing_timing_points** option is also set.

-rise_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to rising signals at the trigger object.

-fall_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to falling signals at the trigger object.

-setup

Indicates that the check value is for setup checking only.

-hold

Indicates that the check value is for hold checking only. If neither of the **-setup** or **-hold** options is specified, the value applies to both setup and hold checks.

-continue

Causes NanoTime to continue propagation after it evaluates the timing check. NanoTime usually stops path tracing at the timing endpoint after evaluating the timing check, unless the **create_timing_check -continue** command or the **set_timing_check_attributes -continue** command is used to modify this default behavior. NanoTime sets the **is_continue** attribute to **true** if the associated timing check has a **-continue** option.

-use_existing_timing_points

When this option is used, the timing check creation must refer only to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is true for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **create_timing_check** command has been issued.

check_value

Specifies the setup or hold time for the check.

DESCRIPTION

NanoTime performs setup and hold checking at structures that it recognizes such as latches, RAM cells, and precharge gates. In addition to these timing checks, you can manually specify a setup or hold timing check between any clock port or pin and any data port or pin by using the **create_timing_check** command.

The command specifies a "from" object, a "to" object, and the setup or hold time. The "from" object is a port or pin where there is a clock signal, and is also known as the reference pin. The "to" object is a port or pin where there is a data signal, and is also known as the checked pin. NanoTime performs the timing checks between the specified "from" and "to" points.

For a setup check, NanoTime verifies that the data signal changes by at least a specified amount of time before a clock transition. For a hold check, NanoTime verifies that the data signal remains stable for at least the specified amount of time after a clock transition.

If a setup violation occurs, NanoTime adjusts the data arrival time backward to the passing arrival time and continues path tracing forward. If a hold violation occurs, NanoTime stops path tracing.

A timing check specified by this command is performed in addition to any other checks already being performed between the "from" and "to" points. It does not replace any existing checks. After running the **check_design** command, you can modify a timing check with the **set_timing_check_attributes** command. You can remove a timing check with the **remove_timing_check** command.

A timing check specified by this command is a flip-flop style edge triggered timing constraint. It is different than a timing check at a non-transparent gated clock specified by the **create_gated_clock_timing_check** command. Though no warning message is issued by NanoTime, intermingling edge-triggered and non-transparent timing checks at a timing point can lead to unexpected results.

COMMAND PHASING

In a typical NanoTime flow,the **create_timing_check** command must be executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands).

If you use the command after the **check_design** command, NanoTime ordinarily transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command. However, if you use the **-use_existing_timing_points** option, you can use the **create_timing_check** command after the **check_design** command has been issued.

EXAMPLES

There are two input ports, CLK (a rising-edge clock signal) and DATA1 (a data signal). The data signal must be stable (either high or low) at least 0.5 time units before a rising transition at CLK, and must remain stable for at least 0.7 time units after the rising transition at CLK. To specify these checks:

```
nt_shell> create_timing_check \
          -rise_from [get_ports CLK] \
          -to [get_ports DATA1] -setup 0.5

nt_shell> create_timing_check \
          -rise_from [get_ports CLK] \
          -to [get_ports DATA1] -hold 0.7
```

.

SEE ALSO

check_design(2)
remove_timing_check(2)
report_paths(2)
set_data_check(2)
create_gated_clock_timing_check(2)
set_timing_check_attributes(2)
trace_paths(2)

create_topology_library

Creates a new topology library; used only in a .ntlib file in the topology library directory.

SYNTAX

```
status create_topology_library
   [-version version_string]
   [-description description_string]
```

Data Types

```
   version_string          string
   description_string      string
```

ARGUMENTS

-version *version_string*

Specifies a string used to reference the version number of the topology library.

-description *description_string*

Specifies a string used to reference the contents of the topology library.

DESCRIPTION

This command creates a topology library and specifies the version and description strings for the library. The command is used only in the .ntlib file in the topology library directory, not at the nt_shell prompt.

When you read in a library with the **read_topology_library** command, NanoTime executes the **create_topology_library** command in the .ntlib file in the library directory. The **create_topology_library** command creates the topology library and assigns a user-specified version string and description string to the library. The two strings do not affect topology recognition.

COMMAND PHASING

The **create_topology_library** command is not phase-restricted.

EXAMPLES

You create a topology library directory named topo_lib1. The directory contains a file named topo_lib1.ntlib. The file contains the following command:

```
create_topology_library \
  [-version V20070126] \
  [-description "User Topo Library 1"]
```

SEE ALSO

```
read_topology_library(2)
list_topology_library(2)
report_topology_library(2)
```

current_design

Sets or reports the name of the current design in NanoTime.

SYNTAX

```
status current_design
   [design_name]
```

Data Types

```
   design_name          string
```

ARGUMENTS

design_name

Specifies the working design on which many NanoTime commands operate. It must be the name of a loaded design.

DESCRIPTION

The **current_design** command sets or reports the name of the current design. If a design name is specified, it must be the name of a loaded design. If no design name is specified, the command returns a collection containing the current design.

To display a list of all designs currently loaded and available in NanoTime, use the **list_designs** command.

The combination of the current design and current instance defines the context for many NanoTime commands.

COMMAND PHASING

The **current_design** command is not phase-restricted.

EXAMPLES

The following example uses the **current_design** command to show the current context and to change the context from one design to another.

```
nt_shell> current_design
{"TOP"}

nt_shell> list_designs

Design Registry:
    ADDER       /designs/dbs/my_design.db:ADDER
    FULL_ADDER        /designs/dbs/my_design.db:FULL_ADDER
    FULL_SUBTRACTOR   /designs/dbs/my_design.db:FULL_SUBTRACTOR
    HALF_ADDER        /designs/dbs/my_design.db:HALF_ADDER
    HALF_SUBTRACTOR   /designs/dbs/my_design.db:HALF_SUBTRACTOR
    SUBTRACTOR        /designs/dbs/my_design.db:SUBTRACTOR
 *  TOP         /designs/dbs/my_design.db:TOP

nt_shell> current_design ADDER
{"ADDER"}

nt_shell> current_design
{"ADDER"}
```

SEE ALSO

current_instance(2)
list_designs(2)

current_instance

Sets the working instance object in NanoTime and enables other commands to be used relative to a specific instance in the design hierarchy.

SYNTAX

```
status current_instance
   [instance]
```

Data Types

```
   instance        string
```

ARGUMENTS

instance

Specifies the working instance in NanoTime.

DESCRIPTION

The **current_instance** command sets the working instance in NanoTime to an instance (cell) in the current level of the design hierarchy. You can view a list of cells within the current instance by using the **report_cell** command. The combination of the current design and current instance defines the context for many NanoTime commands.

The **current_design** command changes the working design, then sets the current instance to the top level of the new current design. The **current_instance** command sets the current instance to a cell within the current instance, in the same way that the UNIX **cd** command sets the current directory to a directory within the current directory.

The **current_instance** can operate with a variety of arguments:

- If no argument is specified, the focus is returned to the top level of the hierarchy.

- If the specified instance name is the name of a hierarchical cell at the current level of hierarchy, the current instance is moved down to that level of the design hierarchy.
- If the specified instance name is ".", the current instance is returned and no change is made.
- If the specified instance name is "..", the current instance is moved up one level in the design hierarchy.
- If the specified instance name begins with "/", both the current instance and current design are set.
- Multiple levels of hierarchy can be traversed in a single **current_instance** command by separating multiple cell names with the hierarchy separator character. For example, the **current_instance U1.U2** command sets the current instance down two levels of hierarchy.

By default, the hierarchy separator character in NanoTime is the period character. If you want to use a single period or double period to represent the current or higher-level instance, then you need to set the hierarchy separator character to a different character such as a slash character. This can be done using either of the following methods (using a command or setting a variable):

```
nt_shell> set_hierarchy_separator /

nt_shell> set hierarchy_separator /
```

COMMAND PHASING

The **current_instance** command is not phase-restricted.

EXAMPLES

The following example uses **current_instance** to move up and down the design hierarchy.

```
nt_shell> current_design TOP
{"TOP"}

nt_shell> current_instance U1
U1

nt_shell> set hierarchy_separator /
/

nt_shell> current_instance "."
U1

nt_shell> current_instance U3
U1/U3

nt_shell> current_instance "../U4"
U1/U4

nt_shell> current_instance
Current instance is the top-level of design 'TOP'.
```

The following example uses **current_instance** to go to an instance of another design. The current design

is set by the new design, whose name is the name after the first slash of the given instance name.

```
nt_shell> current_design
{"TOP"}

nt_shell> current_instance U1
U1

nt_shell> current_instance "/TOP/U2"
U2

nt_shell> current_instance "/ALARM_BLOCK/U6"
U6

nt_shell> current_design
{"ALARM_BLOCK"}
```

SEE ALSO

```
all_instances(2)
current_design(2)
list_designs(2)
report_cell(2)
set_hierarchy_separator(2)
hierarchy_separator(3)
```

date

Returns a string containing the current date and time.

SYNTAX

```
string date
```

DESCRIPTION

The **date** command generates a string containing the current date and time, and returns that string as the result of the command. The format is fixed as follows:

```
ddd mmm nn hh:mm:ss yyyy
```

Where:

```
ddd is an abbreviation for the day
mmm is an abbreviation for the month
nn is the day number
hh is the hour number (24 hour system)
mm is the minute number
ss is the second number
yyyy is the year
```

The **date** command is useful because it is native. It does not fork a process. On some operating systems, when the process becomes large, no further processes can be forked from it. With it, there is no need to call the operating system with **exec** to ask for the date and time.

EXAMPLES

The following command prints the date.

```
prompt> echo "Date and time: [date]"
Date and time: Thu Dec  9 17:29:51 1999
```

SEE ALSO

exec(2)

define_proc_attributes

Defines attributes of a Tcl procedure, including an information string for help, a command group, a set of argument descriptions for help, and so on. The command returns the empty string.

SYNTAX

```
string define_proc_attributes
   proc_name
   [-info info_text]
   [-define_args arg_defs]
   [-define_arg_groups group_defs]
   [-command_group group_name]
   [-return type_name]
   [-hide_body]
   [-hidden]
   [-dont_abbrev]
   [-permanent]
```

Data Types

```
   proc_name      string
   info_text      string
   arg_defs       list
   group_defs     list
   group_name     string
   type_name      string
```

ARGUMENTS

proc_name

Specifies the name of the existing procedure.

-info *info_text*

Provides a help string for the procedure. This is printed by the **help** command when you request help for the procedure. If you do not specify *info_text*, the default is "Procedure".

-define_args *arg_defs*

Defines each possible procedure argument for use with **help -verbose**. This is a list of lists where each list element defines one argument.

-define_arg_groups ***group_defs***

Defines argument checking groups. These groups are checked for you in parse_proc_arguments. each list element defines one group

-command_group ***group_name***

Defines the command group for the procedure. By default, procedures are placed in the "Procedures" command group.

-return ***type_name***

Specifies the type of value returned by this proc. Any value may be specified. Some applications use this information for automatically generated dialogs etc. Please see the type_name option attribute described below.

-hide_body

Hides the body of the procedure from **info body**.

-hidden

Hides the procedure from **help** and **info proc**.

-dont_abbrev

Specifies that the procedure can never be abbreviated. By default, procedures can be abbreviated, subject to the value of the **sh_command_abbrev_mode** variable.

-permanent

Defines the procedure as permanent. You cannot modify permanent procedures in any way, so use this option carefully.

-deprecated

Defines the procedure as deprecated i.e. it should no longer be called but is still available.

-obsolete

Defines the procedure as obsolete i.e. it used to exist but can no longer be called.

DESCRIPTION

The **define_proc_attributes** command associates attributes with a Tcl procedure. These attributes are used to define help for the procedure, locate it in a particular command group, and protect it.

When a procedure is created with the **proc** command, it is placed in the Procedures command group. There is no help text for its arguments. You can view the body of the procedure with **info body**, and you can modify the procedure and its attributes. The **define_proc_attributes** command allows you to change these aspects of a procedure.

Note that the arguments to Tcl procedures are all named, positional arguments. They can be programmed with default values, and there can be optional arguments by using the special argument name *args*. The

define_proc_attributes command does not relate the information that you enter for argument definitions with **-define_args** to the actual argument names. If you are describing anything other than positional arguments, it is expected that you are also using **parse_proc_arguments** to validate and extract your arguments.

The *info_text* is displayed when you use the **help** command on the procedure.

Use **-define_args** to define help text and constraints for individual arguments. This makes the help text for the procedure look like the help for an application command. The value for **-define_args** is a list of lists. Each element has the following format:

```
arg_name option_help value_help data_type attributes
```

The elements specify the following information:

- *arg_name* is the name of the argument.
- *option_help* is a short description of the argument.
- *value_help* is the argument name for positional arguments, or a one word description for dash options. It has no meaning for a Boolean option.
- *data_type* is optional and is used for option validation. The *data_type* can be any of: string, list, boolean, int, float, or one_of_string. The default is string.
- *attributes* is optional and is used for option validation. The *attributes* is a list that can have any of the following entries:
    - "required" - This argument must be specified. This attribute is mutually exclusive with optional.
    - "optional" - Specifying this argument is optional. This attribute is mutually exclusive with "required."
    - "value_help" - Indicates that the valid values for a one_of_string argument should be listed whenever argument help is shown.
    - "values {<list of allowable values>}" - If the argument type is one_of_string, you must specify the "values" attribute.
    - "type_name <name>" - Give a descriptive name to the type that this argument supports. Some applications may use this information to provide features for automatically generated dialogs, etc. Please see product documentation for details. This attribute is not supported on boolean options.
    - "merge_duplicates" - When this option appears more than once in a command, its values are concatenated into a list of values. The default behavior is that the right-most value for the option specified is used.
    - "remainder" - Specifies that any additional positional arguments should be returned in this option. This option is only valid for string option types, and by default the option is optional. You can require at least one item to be specified by also including the required option.
    - "deprecated" - Specifying this option is deprecated i.e. it should no longer be used but is still available. A warning will be output if this option is specified. This attribute cannot be combined with obsolete or required.
    - "obsolete" - Specifying this option is obsolete i.e. it used to exist but can no longer be used. A

warning will be output if this option is specified. This attribute cannot be combined with deprecated or required.

- "min_value" <value> - Specify the minimum value for this option. This attribute is only valid for integer and float types.
- "max_value" <value> - Specify the maximum value for this option. This attribute is only valid for integer and float types.
- "default" <value> - Specify the default value for this option. This attribute is only valid for string, integer and float option types. If the user does not specify this option when invoking the command this default value will be automatically passed to the associated tcl procedure.

The default for *attributes* is "required."

Use the **-define_arg_groups** to define argument checking groups. The format of this option is a list where each element in the list defines an option group. Each element has the following format:

{<type> {<opt1> <opt2> ...} [<attributes>]}

The types of groups are

- *"exclusive"* Only one option in an exclusive group is allowed. All the options in the group must have the same required/optional status. This group can contain any number of options.
- *"together"* If the first option in the group is specified then the second argument is also required. This type of group can contain at most two options.
- \fi"related" These options are related to each other. An automatic dialog builder for this command may try to group these options together.

The supported attributes are:

- *{"label" <text>}* An optional label text to identify this group. The label may be used by an application to automatically build a grouping in a generated dialog.
- *"bidirectional"* This is only valid for a together group. It means that both options must be specified together.

Change the command group of the procedure using the **-command_group** command. Protect the contents of the procedure from being viewed by using **-hide_body**. Prevent further modifications to the procedure by using **-permanent**. Prevent abbreviation of the procedure by using **-dont_abbrev**.

EXAMPLES

The following procedure adds two numbers together and returns the sum. For demonstration purposes, unused arguments are defined.

```
prompt> proc plus {a b} { return [expr $a + $b]}

prompt> define_proc_attributes plus -info "Add two numbers" \
? -define_args {
  {a "first addend" a string required}
```

```
  {b "second addend" b string required}
  {"-verbose" "issue a message" "" boolean optional}}

prompt> help -verbose plus
Usage: plus    # Add two numbers
    [-verbose]         (issue a message)
    a                  (first addend)
    b                  (second addend)

prompt> plus 5 6
11
```

In the following example, the argHandler procedure accepts an optional argument of each type supported by **define_proc_attributes**, then displays the options and values received. Note the only one of -Int, -Float, or -Bool may be specified to the command:

```
proc argHandler {args} {
  parse_proc_arguments -args $args results
  foreach argname [array names results] {
    echo $argname = $results($argname)
  }
}

define_proc_attributes argHandler \
  -info "Arguments processor" \
  -define_args {
     {-Oos "oos help"       AnOos   one_of_string
        {required value_help {values {a b}}}}
     {-Int "int help"       AnInt   int     optional}
     {-Float "float help"   AFloat  float   optional}
     {-Bool "bool help"     ""      boolean optional}
     {-String "string help" AString string  optional}
     {-List "list help"     AList   list    optional}
     {-IDup "int dup help"  AIDup   int     {optional merge_duplicates}}
   } \
  -define_arg_groups {
     {exclusive {-Int -Float -Bool}}
    }
```

SEE ALSO

help(2)
info(2)
parse_proc_arguments(2)
proc(2)
sh_command_abbrev_mode(3)

echo

Echos arguments to standard output.

SYNTAX

```
string echo
   [-n]
   [arguments]
```

Data Types

arguments	string

ARGUMENTS

-n

Suppresses the new line. By default, **echo** adds a new line.

arguments

Specifies the arguments to be printed.

DESCRIPTION

The **echo** command prints out the value of the given arguments. Each of the arguments are separated by a space. The line is normally terminated with a new line, but if the **-n** option is specified, multiple **echo** command outputs are printed onto the same output line.

The **echo** command is used to print out the value of variables and expressions in addition to text strings. The output from **echo** can be redirected using the **>** and **>>** operators.

EXAMPLES

The following are examples of using the **echo** command:

```
prompt> echo
"Running version" $sh_product_version
Running version v3.0a

prompt> echo -n "Printing to" [expr (3 - 2)] > foo
prompt> echo " line." >> foo
prompt> sh cat foo
Printing to 1 line.
```

SEE ALSO

sh(2)

erase_clock_gate

Removes recognition of a previously recognized clock gate structure in the design.

SYNTAX

```
status erase_clock_gate
   -output net
```

Data Types

```
   net          string
```

ARGUMENTS

-output *net*

Removes recognition of a clock gate structure annotated on this net.

DESCRIPTION

NanoTime can automatically recognize clock gating structures in the design, where a clock signal is enabled or disabled by one or more control signals. In addition, you can manually specify clock gating structures with the **mark_clock_gate** command. NanoTime performs timing checks on these structures to ensure that the enable signals arrive before the applicable clock edges.

The **erase_clock_gate** command removes recognition of existing clock gating structures, either manually marked by the **mark_clock_gate** command or automatically recognized at the **match_topology** command. You must use the **-output** option to specify the output net of the clock gating structure. This causes NanoTime to skip clock gating timing checks at that net. The command does not prevent future recognition of clock gating structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for clock-gating structures automatically.

COMMAND PHASING

The **erase_clock_gate** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of any clock gating structure having an output at net N1. This prevents NanoTime from performing clock gating timing checks at that net.

```
nt_shell> erase_clock_gate -output N1
```

SEE ALSO

```
mark_clock_gate(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_clock_network

Cancels the effects of the **mark_clock_network** command for specified objects.

SYNTAX

```
status erase_clock_network
   -force_propagation | -stop_propagation | -no_pulse | -end_dcs
   object_list
```

Data Types

```
   object_list     list
```

ARGUMENTS

-force_propagation

Erases "force clock propagation" markings on the specified objects.

-stop_propagation

Erases "stop clock propagation" markings on the specified objects.

-no_pulse

Erases "no pulse properties" markings on the specified objects.

-end_dcs

Erases "end DCS" markings on the specified objects.

object_list

Specifies the list of ports, pins, and nets from which to remove clock propagation markings.

DESCRIPTION

The **erase_clock_network** command cancels the effects of the **mark_clock_network** command for specified ports, pins, and nets. Using the **erase_clock_network** command restores the default clock propagation behavior for the objects in the argument list. You must use exactly one of the options with this command, along with an object list.

COMMAND PHASING

The **erase_clock_network** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes any "force clock propagation" markings on ports, pins, or nets named N1 or N2.

```
nt_shell> erase_clock_network -force_propagation { N1 N2 }
```

The following command removes any "stop clock propagation" markings on ports, pins, or nets named N3 or N4.

```
nt_shell> erase_clock_network -stop_propagation { N3 N4 }
```

SEE ALSO

```
mark_clock_network(2)
match_topology(2)
topo_match_topology_reset_clock_and_topology(3)
```

erase_cross_coupled

Removes recognition of a previously specified cross-coupled structure in the design.

SYNTAX

```
status erase_cross_coupled
   -output net
```

Data Types

```
   net       string
```

ARGUMENTS

-output *net*

Removes recognition of a cross-coupled structure annotated on this net.

DESCRIPTION

NanoTime can automatically recognize cross-coupled topology structures in the design. The **erase_cross_coupled** command erases a previously recognized cross-coupled structure. You must use the **-output** option to specify the output net of the structure. The command does not prevent future recognition of cross-coupled structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for cross-coupled structures automatically.

COMMAND PHASING

The **erase_cross_coupled** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command erases a cross-coupled structure that has an output net N1.

```
nt_shell> erase_cross_coupled -output N1
```

SEE ALSO

```
erase_topology(2)
report_topology(2)
match_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_cross_coupled_pmos

Removes recognition of a previously marked cross-coupled PMOS structure in the design.

SYNTAX

```
status erase_cross_coupled_pmos
   -transistors pmos_devices
```

Data Types

```
   pmos_devices       list
```

ARGUMENTS

-transistors *pmos_devices*

Identifies the two transistors that form a cross-coupled PMOS element.

DESCRIPTION

This command erases a previously marked cross-coupled PMOS transistor structure. The devices must be PMOS devices that were specified using the **mark_cross_coupled_pmos** command.

NanoTime issues an error message if the structure does not conform to a conventional cross-coupled PMOS topology.

COMMAND PHASING

The **erase_cross_coupled_pmos** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase

topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command erases transistor instances MP1 and MP2 as a cross-coupled PMOS transistor structure.

```
nt_shell> erase_cross_coupled_pmos -transistors {MP1 MP2}
```

SEE ALSO

mark_cross_coupled_pmos(2)
report_topology(2)
match_topology(2)

erase_differential_synchronizer

Removes recognition of a previously recognized differential synchronizer structure in the design.

SYNTAX

```
status erase_differential_synchronizer
   -transistors transistor_list
```

Data Types

```
   transistor_list     list
```

ARGUMENTS

-transistors *transistor_list*

Specifies a list of transistors that belong to the differential synchronizer being removed.

DESCRIPTION

NanoTime can automatically recognize differential synchronizer circuits between two nets that form a differential net pair. In addition, you can manually specify a differential synchronizer using the **mark_differential_synchronizer** command.

The **erase_differential_synchronizer** command removes recognition of an existing synchronizer structure, either manually marked by the **mark_differential_synchronizer** command or automatically recognized at the **match_topology** command. You must use the **-transistors** option to specify at least one of the transistors in the synchronizer being removed; you do not need to list all of the transistors. The command does not prevent future recognition of differential synchronizer structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for differential synchronizer structures automatically.

COMMAND PHASING

The **erase_differential_synchronizer** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following **erase_differential_synchronizer** command removes the differential synchronizer that includes transistor MN1.

```
nt_shell> erase_differential_synchronizer -transistors  "MN1"
```

SEE ALSO

```
mark_differential_synchronizer(2)
report_topology(2)
match_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_feedback

Removes recognition of a previously recognized feedback transistor in the design.

SYNTAX

```
status erase_feedback
   -transistor element
```

Data Types

```
   element      string
```

ARGUMENTS

-transistor *element*

Specifies a transistor in the design that should not be considered to be a feedback transistor.

DESCRIPTION

NanoTime automatically recognizes NMOS and PMOS feedback transistors connected between the input and output of an inverter. In addition, you can manually specify that a transistor is a feedback transistor by using the **mark_feedback** command. NanoTime removes feedback transistors from path searches to prevent looping through the circuit.

The **erase_feedback** command removes recognition of an existing feedback transistor, either manually marked by the **mark_feedback** command or automatically recognized at the **match_topology** command. You must use the **-transistor** option to specify the transistor instance. This removes the feedback marking and causes NanoTime to include the transistor in path searches. The command does not prevent future recognition of feedback devices.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for feedback transistors automatically.

COMMAND PHASING

The **erase_feedback** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of transistor MN1 as a feedback transistor.

```
nt_shell> erase_feedback -transistor MN1
```

SEE ALSO

```
mark_feedback(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_flip_flop

Removes recognition of a previously recognized flip-flop at a specified location in the design.

SYNTAX

```
status erase_flip_flop
   -slave_latch slave_object
```

Data Types

```
   slave_object        string
```

ARGUMENTS

-slave_latch *slave_object*

Identifies the latch topology name or the latch net of the slave latch previously identified or marked as the slave of the flip-flop structure.

DESCRIPTION

NanoTime can automatically recognize a flip-flop structure when two latches are connected together sequentially with both latches clocked by the same clock, but with an inverted phase relationship. The first latch is the master, which must have a transparent path to the second latch. The second latch is the slave, which is evaluated as a nontransparent device. The clock that drives the slave must be an inverted version of the clock that drives the master.

The **erase_flip_flop** command removes recognition of an existing flip-flop structure, either manually marked by the **mark_flip_flop** command or automatically recognized at the **match_topology** command. You must use the **-slave_latch** option to specify the slave latch net or the name of the slave latch topology. Then NanoTime no longer considers the structure a flip-flop and the slave latch is set to transparent. The command does not prevent future recognition of flip-flop structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for flip-flop

structures automatically.

COMMAND PHASING

The **erase_flip_flop** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of the latch whose slave latch node is net N1.

```
nt_shell> erase_flip_flop -slave_latch N1
```

SEE ALSO

```
mark_flip_flop(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_instance

Removes path tracing instructions previously set with the **mark_instance** command.

SYNTAX

```
status erase_instance
   [-dont_search_thru_gate]
   [-dont_search_thru_channel]
   [-dont_include_for_side_branch]
   [-exclude_from_dcs]
   elements
```

Data Types

```
   elements       list
```

ARGUMENTS

-dont_search_thru_gate

Allows path tracing through the gate pin of the transistor.

-dont_search_thru_channel

Allows path tracing through the source or drain pin of the transistor.

-dont_include_for_side_branch

Allows the transistor to be included as a side branch.

-exclude_from_dcs

Allow the transistor's channel-connected block as well as its transitive fanout to be included in the dynamic clock simulation region.

elements

The list of transistor elements in the design from which to remove path tracing settings.

DESCRIPTION

The **erase_instance** command cancels the effects of the **mark_instance** command for specified transistors. Using the **erase_instance** command restores the default path searching behavior for the objects in the argument list. You can use more than one option with this command, along with an object list.

If you do not specify any options, nothing is removed and the command has no effect.

COMMAND PHASING

The **erase_instance** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command cancels a **-dont_search_thru_gate** setting on transistor instance m1, which restores the default behavior and allows a path search to go through the gate of the transistor.

```
nt_shell> erase_instance -dont_search_thru_gate m1
```

The following command removes all **mark_instance** settings on transistors in cell Xxor7.X0, which restores the default path tracing behavior for those transistors.

```
nt_shell> erase_instance \
          -dont_search_thru_gate \
          -dont_search_thru_channel \
          -dont_include_for_side_branch \
          -exclude_from_dcs \
          [get cells Xxor7.X0.*]
```

SEE ALSO

```
mark_instance(2)
trace_paths(2)
```

erase_inverter

Removes recognition of previously recognized inverter(s).

SYNTAX

```
status erase_inverter
   -transistors elements
```

Data Types

```
   elements         list
```

ARGUMENTS

-transistors *elements*

Removes recognition of inverter(s) containing these transistors.

DESCRIPTION

NanoTime automatically recognizes the standard CMOS inverter structure consisting of one PMOS transistor and one NMOS transistor. In addition, you can manually specify an inverter with the **mark_inverter** command. NanoTime uses this information to propagate clocks and to trace data signals through the design.

The **erase_inverter** command removes recognition of an existing inverter structure, either manually marked by the **mark_inverter** command or automatically recognized at the **match_topology** command. You must use the **-transistors** option to specify any one transistor instance within the recognized inverter structure. Then NanoTime no longer considers the structure to be an inverter. A list of transistors can be provided to remove recognition of multiple inverters at once, which can be significantly more efficient than running this command one inverter at a time. The command does not prevent future recognition of inverter structures.

COMMAND PHASING

The **erase_inverter** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of the inverter structure containing transistor MN1.

```
nt_shell> erase_inverter -transistors MN1
```

SEE ALSO

```
mark_inverter(2)
match_topology(2)
report_topology(2)
topo_match_topology_reset_clock_and_topology(3)
```

erase_latch

Removes recognition of a previously recognized latch at a specified latch net in the design.

SYNTAX

```
status erase_latch
   -latch_net net
```

Data Types

```
   net              string
```

ARGUMENTS

-latch_net *net*

Specifies the latch net of the latch to be removed.

DESCRIPTION

NanoTime can automatically recognize latch structures of various types, based on the meeting of clock and data signals where inverters, NAND and NOR gates, or three-state elements are arranged in a feedback configuration. In addition, you can manually specify latches with the **mark_latch** command. NanoTime performs latch timing checks on these structures to ensure that the setup and hold constraints are satisfied.

The **erase_latch** command removes recognition of an existing latch structure, either manually marked by the **mark_latch** command or automatically recognized at the **match_topology** command. You must use the **-latch_net** option to specify the name of the net that is the latch node. Then NanoTime no longer considers the structure to be a latch and does not perform latch timing checks. The command does not prevent future recognition of latch structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for latch structures automatically.

COMMAND PHASING

The **erase_latch** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of the latch whose latch node is net N1.

```
nt_shell> erase_latch -latch_net N1
```

SEE ALSO

```
mark_latch(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_memory_bitline

Removes recognition of previously recognized memory bitlines and their complements in the design.

SYNTAX

```
status erase_memory_bitline
   bitlines
   -memory memory_name
```

Data Types

```
   bitlines        list
   memory_name     string
```

ARGUMENTS

bitlines

A list of memory bitlines to erase.

-memory ***memory_name***

Specifies the name of the memory structure that the bitlines are associated with, corresponding to the **-name** argument of the **create_memory** command.

DESCRIPTION

In the **skip_array** mode analysis, the user can define two objects to be a memory bitline and its complement with the **mark_memory_bitline** command.

The **erase_memory_bitline** command removes recognition of every bitline object in the **bitlines** list and its complement associated with the named memory specified in the **-memory** argument. This command assumes that those bitlines and their complements are previously defined by the **mark_memory_bitline** command.

Like the **mark_memory_bitline** command, the **erase_memory_bitline** command can be executed only in the memory **skip_array** mode analysis. Nanotime ignores this command in a different memory mode of operation.

This command may be used inside a foreach_match block to efficiently erase all bitlines in the array.

COMMAND PHASING

The **erase_memory_bitline** command must be executed between the **create_memory** and **check_topology** commands.

EXAMPLES

The following command removes recognition of memory bitlines, bit0 and bit1, and their complements that are associated with the named memory, sa_mem.

```
nt_shell> erase_memory_bitline {bit0 bit1} -memory sa_mem
```

SEE ALSO

```
create_memory(2)
report_memory(2)
report_bitline(2)
mark_memory_bitline(2)
```

erase_memory_precharge

Removes recognition of previously recognized memory precharge structures in the design.

SYNTAX

```
status erase_memory_precharge
   -evaluate_nets net_list
```

Data Types

```
   net_list      list
```

ARGUMENTS

-evaluate_nets *net_list*

The list of nets that determines which structures to erase.

DESCRIPTION

NanoTime for memories can automatically recognize memory precharge structures in the design. In addition, you can manually specify memory precharge structures with the **mark_memory_precharge** command. NanoTime for memories uses this information to configure memory bitline simulations.

The **erase_memory_precharge** command removes recognition of existing memory precharge structures, either manually marked by the **mark_memory_precharge** command or automatically recognized at the **match_topology** command. You must use the **-evaluate_nets** option to identify the structures to erase. Each net in the list is processed; any memory precharge structure that has an evaluate net at one of the specified nets is erased. The command does not prevent future recognition of memory precharge structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for memory precharge structures automatically.

COMMAND PHASING

The **erase_memory_precharge** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following **erase_memory_precharge** command removes recognition of a memory precharge structure that has an evaluate net N1.

```
nt_shell> mark_memory_precharge -evaluate_net N1
```

SEE ALSO

```
mark_memory_precharge(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_memory_write_circuit

Removes recognition of previously recognized memory write circuit structures in the design.

SYNTAX

```
status erase_memory_write_circuit
   -outputs net_list
```

Data Types

```
   net_list     list
```

ARGUMENTS

-outputs *net_list*

The list of nets that determines which structures to erase.

DESCRIPTION

NanoTime for memories recognizes memory write circuit structures manually specified with the **mark_memory_write_circuit** command. NanoTime uses this information to configure write mode memory simulations.

The **erase_memory_write_circuit** command removes recognition of existing memory write circuit structures identified by **mark_memory_write_circuit** commands. You must use the **-outputs** option to specify one of the output nets for the structure to be erased. When the argument list contains multiple nets, each net in the list is processed for memory write circuit structures to erase.

COMMAND PHASING

The **erase_memory_write_circuit** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the

check_topology command.

EXAMPLES

The following example first uses the **mark_memory_write_circuit** command in a loop to mark some memory write circuits. The **report_topology** command reports the flattened names of the topologies. Finally, the **erase_memory_write_circuit** command removes recognition of a memory write circuit structure that has output net Xmem0/bit0.

```
nt_shell> foreach_match wdrvr_pwdn -command {
     mark_memory_write_circuit -outputs bit0 -transistors Mn0
       -enable Mn0/g
        }

nt_shell> report_topology -verbose -structure_type memory_write_circuit

...
Attributes:
      u - User Defined
      C - Common Topology Library Defined
      U - User Topology Library Defined
      G - Global Topology Library Defined

Output Nets    Transistors        Enable Pin          Input Nets   Attrs
-------------- ------------------ ------------------ ------------ -----
Xmem0/bit0     Xmem0/Xwdrv/Mn0    Xmem0/Xwdrv/Mn0/g                  u
...

nt_shell> erase_memory_write_circuit -outputs Xmem0/bit0
```

SEE ALSO

```
mark_memory_write_circuit(2)
match_topology(2)
report_topology(2)
```

erase_mux

Removes recognition of a previously recognized multiplexer structure in the design.

SYNTAX

```
status erase_mux
   -output_net net
   -select_pins pin_list
```

Data Types

```
   net             string
   pin_list        list
```

ARGUMENTS

-output_net *net*

Specifies the output net of the mux.

-select_pins *pin_list*

Specifies the select pins of the mux.

DESCRIPTION

NanoTime can automatically recognize mux (multiplexer) structures based on the presence of parallel pass gates or transmission gates that are connected together at the source or drain. In addition, you can manually specify a multiplexer structure with the **mark_mux** command. NanoTime uses this information to prevent the tracing of false paths.

The **erase_mux** command removes recognition of existing mux structures, either manually marked by the **mark_mux** command or automatically recognized at the **match_topology** command. You must use the **-output_net** option to specify the output net of the multiplexer and the **-select_pins** option to specify the names of the selection input pins. Then NanoTime no longer considers the structure to be a

multiplexer. The command does not prevent future recognition of mux structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for multiplexer topology structures automatically.

COMMAND PHASING

The **erase_mux** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of a mux having an output at net N1 and select pins IN1 and IN2.

```
nt_shell> erase_mux -output_net N1 -select_pins { IN1 IN2 }
```

SEE ALSO

```
mark_mux(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_net

Removes instructions previously marked on a net with the **mark_net** command.

SYNTAX

```
status erase_net
   [-precharge]
   [-check_miller_direction]
   [-ignore_keep_within]
   [-clock_gate_checking force_strict | non_strict]
   [-differential_cross_coupled]
   [-differential_override pins]
   [-transistor_pin_load_model]
   nets
```

Data Types

```
   pins            list
   nets            list
```

ARGUMENTS

-precharge

Removes recognition of the specified nets as precharge evaluate nets.

-check_miller_direction

Prevents direction checking when active Miller loads are being used.

-ignore_keep_within

Removes the effect of ignoring the margin specified by the **trace_paths -keep_paths_within** command.

-clock_gate_checking force_strict | non_strict

Removes specification of the type of checking to be carried out for clock gates that drive this net. Valid options are **force_strict** or **non_strict**.

-differential_cross_coupled

Removes the specified differential net from forming a cross-coupled structure with the paired differential net.

-differential_override *pins*

Removes the specified gate pins from overriding the differential behavior of the specified net.

-transistor_pin_load_model

Removes the effect of setting the transistor loading modeling for turned-off transistors which are source or drain connected to the specified nets.

-ignore_high_impedance_pessimism_reduction

Removes the effect of ignoring the high_impedance pessimism reduction attribute.

nets

Specifies the nets in the design from which to remove the markings.

DESCRIPTION

The **mark_net** command lets you specify nets for which to apply special instructions that affect path tracing. Use the **erase_net** command to remove markings previously applied with the **mark_net** command. For more information about the marking options, see the man page for the **mark_net** command.

The **erase_net -precharge** command removes markings applied with the **mark_net -precharge** command.

The **erase_net -check_miller_direction** command removes markings applied with the **mark_net -check_miller_direction** command.

The **erase_net -ignore_keep_within** command removes markings applied with the **mark_net -ignore_keep_within** command.

The **erase_net -clock_gate_checking** command removes markings applied with the **mark_net -clock_gate_checking** command.

The **erase_net -differential_cross_coupled** command removes markings applied with the **mark_net -differential_cross_coupled**, **mark_net -differential_cross_coupled_preset**, or **mark_net -differential_cross_coupled_clear** commands.

The **erase_net -differential_override** command removes markings applied with the **mark_net -differential_override_preset** or **mark_net -differential_override_clear** commands.

The **erase_net -transistor_pin_load_model** command removes markings applied with the **mark_net -transistor_pin_load_model** command.

COMMAND PHASING

The **erase_net** command is typically executed in the "netlist-linked" state (between the **link_design**

command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command removes recognition of net Sn03 as a precharge evaluate net.

```
nt_shell> erase_net -precharge Sn03
```

SEE ALSO

```
mark_net(2)
sim_miller_direction_check(3)
match_topology(2)
```

erase_pin

Removes instructions previously marked on a pin with the **mark_pin** command.

SYNTAX

```
status erase_pin
   [-kill_path_constraints]
   pins
```

Data Types

```
   pins     list
```

ARGUMENTS

-kill_path_constraints

Removes instructions marked on the specified pins to ignore the current logic constraints.

pins

Specifies the pins in the design from which to remove the markings.

DESCRIPTION

The **mark_pin** command marks one or more pins in the design with instructions to ignore the current logic constraints on that pin.

To cancel the effects of the **mark_pin** command, use the **erase_pin** command.

COMMAND PHASING

The **erase_pin** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command removes the ignore-constraints marking on pin Xxor7.A.

```
nt_shell> erase_pin -kill_path_constraints Xxor7.A
```

SEE ALSO

```
mark_pin(2)
match_topology(2)
```

erase_power_switch

Removes recognition of a previously recognized power switch topology in the design.

SYNTAX

```
status erase_power_switch
   -outputs net
```

Data Types

```
   net          list
```

ARGUMENTS

-outputs *net*

Specifies the output net of the power switch structure.

DESCRIPTION

NanoTime can automatically recognize power switch structures in the design, where power switch topologies are structures between true supply rails and virtual supply rails. In addition, you can manually specify power switch structures with the **mark_power_switch** command.

The **erase_power_switch** command removes recognition of existing power switch structures, either manually marked by the **mark_power_switch** command or automatically recognized at the **match_topology** command. You must use the **-outputs** option to specify the output net (the virtual supply net) of the power switch structure.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for power switch structures automatically.

COMMAND PHASING

The **erase_power_switch** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of a power switch having an output at virtual supply net vvdd.

```
nt_shell> erase_power_switch -outputs vvdd
```

SEE ALSO

```
mark_power_switch(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_precharge

Removes recognition of previously recognized domino precharge structures at specified nets in the design.

SYNTAX

```
status erase_precharge
   net_list
```

Data Types

```
   net_list       list
```

ARGUMENTS

net_list

Specifies a list of nets at which to check for precharge structures to erase.

DESCRIPTION

NanoTime can automatically recognize a variety of domino precharge structures, based on the configuration of transistors, clock signals, and data signals. In addition, you can manually specify precharge structures with the **mark_precharge** command. NanoTime uses this information to trace clocks and timing paths.

The **erase_precharge** command removes recognition of existing precharge structures, either manually marked by the **mark_precharge** command or automatically recognized at the **match_topology** command. You must specify a list of nets to be considered as precharge nodes to determine which structures to erase. The command does not prevent future recognition of precharge structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for precharge structures automatically.

COMMAND PHASING

The **erase_precharge** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of a precharge structure that has its precharge node at net N1.

```
nt_shell> erase_precharge N1
```

SEE ALSO

```
mark_precharge(2)
match_topology(2)
report_topology(2)
```

erase_pulldown

Removes recognition of previously recognized pulldown transistors in the design.

SYNTAX

```
status erase_pulldown
   -transistors elements
```

Data Types

```
   elements      list
```

ARGUMENTS

-transistors *element*

Specifies the transistors in the design that should not be considered pulldown transistors.

DESCRIPTION

The **mark_pulldown** command lets you specify the location of a pulldown transistor in the design, where a rising-edge transition cannot be propagated due to the lack of a pullup transistor.

To cancel the effects of the **mark_pulldown** command, use the **erase_pulldown** command. The **-transistors** option specifies the transistors in the design where rising-edge transitions are to be allowed.

COMMAND PHASING

The **erase_pulldown** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase

topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of transistor mn3 as a pulldown transistor.

```
nt_shell> erase_pulldown -transistors mn3
```

SEE ALSO

mark_pulldown(2)
match_topology(2)
report_topology(2)

erase_pullup

Removes recognition of previously recognized pullup transistors in the design.

SYNTAX

```
status erase_pullup
   -transistors elements
```

Data Types

```
   elements       list
```

ARGUMENTS

-transistors *element*

Specifies the transistors in the design that should not be considered to be pullup transistors.

DESCRIPTION

The **mark_pullup** command lets you specify the location of a pullup transistor in the design, where a falling-edge transition cannot be propagated due to the lack of a pulldown transistor.

To cancel the effects of the **mark_pullup** command, use the **erase_pullup** command. The **-transistors** option specifies the transistors in the design where falling-edge transitions are to be allowed.

COMMAND PHASING

The **erase_pullup** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase

topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of transistor mp8 as a pullup transistor.

```
nt_shell> erase_pullup -transistors mp8
```

SEE ALSO

```
mark_pullup(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_ram

Removes recognition of a previously recognized RAM cell in the design.

SYNTAX

```
status erase_ram
   -node net
```

Data Types

```
   net          string
```

ARGUMENTS

-node *net*

Removes recognition of any RAM cell containing the specified net.

DESCRIPTION

NanoTime can automatically recognize a RAM structure where two equal-strength inverters feed back into each other. In addition, you can manually specify RAM structures with the **mark_ram** command. NanoTime terminates a path search when it reaches a RAM cell.

The **erase_ram** command removes recognition of existing RAM structures, either manually marked by the **mark_ram** command or automatically recognized at the **match_topology** command. You must use the **-node** option to specify either input net of the RAM structure to be erased. Then NanoTime no longer considers the structure to be a RAM cell. The command does not prevent future recognition of RAM structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for RAM structures automatically.

COMMAND PHASING

The **erase_ram** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of a RAM cell containing net n48.

```
nt_shell> erase_ram -node n48
```

SEE ALSO

```
mark_ram(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_register_file

Removes recognition of a register file cell in the design.

SYNTAX

```
status erase_register_file
   -net net
```

Data Types

```
   net          string
```

ARGUMENTS

-net *net*

Removes recognition of any register file cell containing the specified net.

DESCRIPTION

NanoTime can automatically recognize a register file structure where two equal-strength inverters feed back into each other and are controlled by two clocks. In addition, you can manually specify register file structures with the **mark_register_file** command. NanoTime terminates a path search when it reaches a register file cell.

The **erase_register_file** command removes recognition of existing register file structures, either manually marked by the **mark_register_file** command or automatically recognized at the **match_topology** command. You must use the **-net** option to specify either input net of the register file structure to be erased. Then NanoTime no longer considers the structure to be a register file. The command does not prevent future recognition of register file structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for register file structures automatically.

COMMAND PHASING

The **erase_register_file** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of a register file cell containing net n48.

```
nt_shell> erase_register_file -net n48
```

SEE ALSO

```
mark_register_file(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_sense_amp

Removes recognition of sense amplifier structures with the specified input nets.

SYNTAX

```
status erase_sense_amp
   -inputs net_list
```

Data Types

```
   net_list    list
```

ARGUMENTS

-inputs *net_list*

Removes recognition of any sense amplifier structure that has one of the specified nets as an input net.

DESCRIPTION

NanoTime for memories can automatically recognize certain sense amplifier structures based on the configuration of transistors, enable signals, inputs and outputs. In addition, you can manually specify sense amplifier structures with the **mark_sense_amp** command. NanoTime uses this information to trace clocks and timing paths through the structure and to apply timing checks.

The **erase_sense_amp** command removes recognition of existing sense amplifier structures, either manually marked by the **mark_sense_amp** command or automatically recognized at the **match_topology** command. You must use the **-inputs** option to specify one or more input nets that are the recognized differential inputs to the sense amplifiers to be erased. The command does not prevent future recognition of sense amplifier structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for sense amplifier structures automatically.

COMMAND PHASING

The **erase_sense_amp** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following **erase_sense_amp** command removes recognition of a sense-amp structure that has input nets bit and bitb.

```
nt_shell> erase_sense_amp -inputs {bit bitb}
```

SEE ALSO

```
mark_sense_amp(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_tgate

Removes recognition of a transmission gate in the design.

SYNTAX

```
status erase_tgate
   -transistor element
```

Data Types

```
   element         string
```

ARGUMENTS

-transistor *element*

Removes recognition of the transmission gate containing the specified transistor.

DESCRIPTION

NanoTime can automatically recognize a transmission gate structure consisting of an NMOS and a PMOS transistor with their sources and drains connected, and with an inverter between the gates of the transistor. In addition, you can manually specify transmission gate structures with the **mark_tgate** command. NanoTime uses information about transmission gates to correctly generate critical paths and to improve delay calculation accuracy.

When the **topo_tgate_mark_all_pairs** variable is set to **true** (the default), the inversion between the gates of a PMOS and an NMOS transistor is not needed for automatic recognition of the transmission gate topology.

The **erase_tgate** command removes recognition of existing transmission gate structures, either manually marked by the **mark_tgate** command or automatically recognized at the **match_topology** command. You must use the **-transistor** option to specify either one of the two pass transistors of the transmission gate to be erased. The command does not prevent future recognition of transmission gate structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for RAM structures automatically.

COMMAND PHASING

The **erase_tgate** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of the transmission gate containing transistor mp18.

```
nt_shell> erase_tgate -transistor mp18
```

SEE ALSO

```
mark_tgate(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
topo_tgate_mark_all_pairs(3)
```

erase_topology

Removes recognition of topology structures obtained with the **get_topology** command.

SYNTAX

```
status erase_topology
   topologies
```

Data Types

```
   topologies          collection
```

ARGUMENTS

topologies

A collection of one or more topologies obtained with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes a variety of topology structures such as latches, muxes, and transmission gates. You can also explicitly identify structures with commands such as **mark_latch**, **mark_mux**, and **mark_tgate**. To remove recognition of structures, you can use commands such as **erase_latch**, **erase_mux**, and **erase_tgate**.

An alternative method of removing recognition of structures is to use the **erase_topology** command in conjunction with the **get_topology** command. The **get_topology** command creates a collection of structures and the **erase_topology** command removes recognition of all the structures in the collection.

Using the **get_topology** and **erase_topology** commands can be easier to use than the **erase_*** commands because of the flexibility and power of the **get_topology** command. For example, you can easily get a collection of all instances of a very specific structure and erase them all at once.

COMMAND PHASING

The **erase_topology** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following example removes recognition of all latch structures in the design.

```
nt_shell> erase_topology [get_topology -all -structure_type latch]
```

The following example does the same thing. First it creates a collection of all latch structures in the design. Then it removes recognition of those structures.

```
nt_shell> set mylatches [get_topology -all -structure_type latch]
nt_shell> erase_topology $mylatches
```

The following example erases recognition of all latches having the structure name "mylatch1".

```
nt_shell> set mycollection [get_topology -all -structure_type latch \
          -filter "structure_name == mylatch1"]
nt_shell> erase_topology $mycollection
```

SEE ALSO

```
get_topology(2)
match_topology(2)
report_topology(2)
```

erase_turnoff

Removes recognition of a three-state turnoff structure in the design.

SYNTAX

```
status erase_turnoff
   -output_net net
```

Data Types

```
   net       string
```

ARGUMENTS

-output_net *net*

Removes recognition of a turnoff structure having the specified output.

DESCRIPTION

NanoTime can automatically recognize a three-state turnoff structure consisting of two PMOS pullup transistors and two NMOS pulldown transistors connected in series, with complementary data and enable signals at the respective PMOS and NMOS transistor gates. In addition, you can manually specify turnoff structures with the **mark_turnoff** command. NanoTime performs contention analysis at each turnoff structure.

The **erase_turnoff** command removes recognition of existing turnoff structures, either manually marked by the **mark_turnoff** command or automatically recognized at the **match_topology** command. You must use the **-output_net** option to specify the net that should not be a turnoff output. Then NanoTime no longer considers the structure to be a turnoff structure. The command does not prevent future recognition of turnoff structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for turnoff structures automatically.

COMMAND PHASING

The **erase_turnoff** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of the turnoff structure having its output at the net N1.

```
nt_shell> erase_turnoff -output_net N1
```

SEE ALSO

```
mark_turnoff(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_weak_pullup

Removes recognition of a weak pullup transistor in the design.

SYNTAX

```
status erase_weak_pullup
   -transistors elements
```

Data Types

```
   elements          list
```

ARGUMENTS

-transistors *elements*

Specifies a transistor in the design that should not be considered a weak pullup transistor.

DESCRIPTION

NanoTime can automatically recognize a weak pullup transistor. The tool recognizes an NMOS transistor as a weak pullup when its gate and drain are connected to Vdd, or a PMOS transistor when its source is connected to Vdd and its gate is connected to ground. In addition, you can manually specify that a transistor is a weak pullup by using the **mark_weak_pullup** command.

The **erase_weak_pullup** command removes recognition of existing weak pullup transistors, either manually marked by the **mark_weak_pullup** command or automatically recognized at the **match_topology** command. You must use the **-transistors** option to specify the transistor instance. This removes the weak pullup marking and causes NanoTime to consider the transistor to be a strong pullup, which stops path tracing through the pullup node. The command does not prevent future recognition of weak pullup transistors.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for weak pullup transistors automatically.

COMMAND PHASING

The **erase_weak_pullup** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of transistors m1 and m2 as weak pullups, causing NanoTime to consider them strong pullups.

```
nt_shell> erase_weak_pullup {m1 m2}
```

SEE ALSO

```
mark_weak_pullup(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

erase_xor

Removes recognition of an XOR gate in the design.

SYNTAX

```
status erase_xor
   -output_net net
   -input_pins pin_list
```

Data Types

```
   net            string
   pin_list       list
```

ARGUMENTS

-output_net *net*

Specifies the output net of the XOR structure to be erased.

-input_pins *pin_list*

Specifies the input pins of the XOR structure to be erased.

DESCRIPTION

NanoTime can automatically recognize a particular 6-transistor implementation of XOR (exclusive OR) and XNOR (exclusive NOR) logic gates. In addition, you can manually specify an XOR structure by using the **mark_xor** command. NanoTime uses this XOR or XNOR information to eliminate false paths for certain logic states.

The **erase_xor** command removes recognition of existing XOR structures, either manually marked by the **mark_xor** command or automatically recognized at the **match_topology** command. You must use the **-input_pins** option to specify the input pins of the structure and the **-output_net** option to specify the output net. The command does not prevent future recognition of XOR structures.

Note that the **topo_auto_search_class** variable controls whether NanoTime searches for XOR structures automatically.

COMMAND PHASING

The **erase_xor** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

Before the first **match_topology** command, you can only use the **erase_*** commands to erase topologies that have been manually marked. After a **match_topology** command, you can use the **erase_*** commands to erase any existing topology.

If you use the **erase_*** commands after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. In this case, NanoTime is likely to rerecognize any automatically recognized structures that you erased. To avoid this, use the **erase_*** commands after the last **match_topology** command in the flow.

EXAMPLES

The following command removes recognition of an XOR structure that has an output net out1 and input pins mn6 and mn7.

```
nt_shell> erase_xor -output_net out1 -input_pins {mn6 mn7}
```

SEE ALSO

```
mark_xor(2)
match_topology(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

error_info

Prints extended information on errors from the last command.

SYNTAX

```
string error_info
```

ARGUMENTS

This **error_info** command has no arguments.

DESCRIPTION

The **error_info** command is used to display information after an error has occurred. Tcl collects information showing the call stack of commands and procedures. When an error occurs, the **error_info** command can help you to focus on the exact line in a block that caused the error.

EXAMPLES

This example shows how **error_info** can be used to trace an error. The error is that the iterator variable "s" is not dereferenced in the 'if' statement. It should be '$s == "a" '.

```
prompt> foreach s $my_list {
?         if { s == "a" } {
?           echo "Found 'a'!"
?         }
?       }
Error: syntax error in expression " s == "a" "
       Use error_info for more info. (CMD-013)
shell> error_info
```

```
Extended error info:
syntax error in expression " s == "a" "
    while executing
"if { s == a } {
     echo "Found 'a'"
}"
    ("foreach" body line 2)
    invoked from within
"foreach s [list a b c] {
   if { s == a } {
     echo "Found 'a'"
}
}"
 -- End Extended Error Info
```

exclude_multi_input_switching

Excludes the specified nets and their channel-connected regions from consideration during timing-based multi-input switching analysis.

SYNTAX

```
status exclude_multi_input_switching
   [-remove]
   nets
```

Data Types

```
   nets      list
```

ARGUMENTS

-remove

Removes the exclusion on the specified nets. In other words, the nets and their channel-connected regions can now be considered for multi-input switching analysis.

-min

Add/Remove the exclusion only for min arrivals.

-max

Add/Remove the exclusion only for max arrivals.

-rise

Add/Remove the exclusion only for rise arrivals.

-fall

Add/Remove the exclusion only for fall arrivals.

nets

Specifies the nets in the design to be excluded.

DESCRIPTION

The **exclude_multi_input_switching** command excludes the channel-connected regions associated with the specified nets from being 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.

Excluding nets might be required for regions of the design where multi-input switching analysis is not appropriate, such as sensitive clock circuits.

Use the **-remove** option to remove the exclusion from specified nets. This makes the nets and their channel-connected regions available for reselection during the iterative analysis.

The impact of the **exclude_multi_input_switching** command can be further limited by the usage of -min/-max and -rise/-fall options.

You can generate a report of the currently active exclusions by using the **report_multi_input_switching** command.

COMMAND PHASING

The **exclude_multi_input_switching** command must be executed before the **check_design** command.

SEE ALSO

```
report_multi_input_switching(2)
timing_enable_multi_input_switching(3)
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)
```

exit

Terminates the application.

SYNTAX

```
integer exit
   [exit_code]
```

Data Types

```
   exit_code      integer
```

ARGUMENTS

exit_code

Specifies the return code to the operating system. The default value is 0.

DESCRIPTION

This command exits from the application. You have the option to specify a code to return to the operating system.

EXAMPLES

The following example exits the current session and returns the code 5 to the operating system. At a UNIX operating system prompt, verify (**echo**) the return code as shown.

```
prompt> exit 5
```

```
% echo $status
5
```

SEE ALSO

quit(2)

extract_model

Generates a timing model for the current design which can be used for analysis at higher levels of the design hierarchy and in other tools such as PrimeTime.

SYNTAX

```
status extract_model
   [-name model_name]
   [-library_elements element_list]
   [-latch_level levels]
   [-ignore_boundary_parasitics]
   [-extract_boundary_parasitics file_format]
   [-test_design]
   [-arc_types arc_list]
   [-ignore_ports ignore_port_list]
   [-extend_inputs extend_input_port_list]
   [-extend_outputs extend_output_port_list]
   [-debug debug_output_file_list]
   [-non_transparent]
   [-ignore_input_to_output_transparency]
   [-combinational_input_to_output_transparency]
   [-keep_paths_within time_value]
   [-npaths number_of_paths]
   [-pbsa]
   [-pocv]
   [-bus]
   [-complete_bus]
   [-mode mode_name]
   [-use_find_path]
   [-find_path_only]
   [-non_sequential_arcs]
   [-conform_arc_types]
   [-context_dependent]
   [-extend_clock_domains]
   [-enable_arc_segmentation]
   [-message_level level]
   [-hspice_timing]
   [-when]
```

Data Types

model_name	string
element_list	list
levels	integer
file_format	string
arc_list	list
ignore_port_list	list
extend_input_port_list	list
extend_output_port_list	list

```
debug_output_file_list      list
time_value                  float
number_of_paths             integer
mode_name                   string
level                       string
```

ARGUMENTS

-name *model_name*

Specifies the file name used as a prefix for generating the model files. It can be an absolute path (beginning with a slash character) or a relative path. Specifying a name is necessary in most situations. If no name is specified, NanoTime builds an in-memory model, without generating any model files, from which you can extract information for building a custom timing model.

-library_elements *element_list*

Specifies the type of library model elements produced in the model. The list can be *nldm* to produce only NLDM modeling data (the default), *{nldm ccs_timing}* to produce both NLDM and CCS timing data, *{nldm pg_pins}* to produce NLDM with power and ground pins, or *{nldm ccs_timing pg_pins}* to produce both NLDM and CCS timing data with power and ground pins. The **ccs_timing** and **pg_pins** arguments cannot be used alone. The list can be *{nldm ccs_timing ccs_noise}* to produce CCS noise data along with NLDM and CCS timing data. The **ccs_noise** argument cannot be used alone.

-latch_level *levels*

Specifies the latch transparency depth (maximum number of successive transparent latches per path) considered during model generation. A setting of 0 results in a black box model that does not consider any latch transparency. The default is the maximum allowed setting, 65535, which causes the model generator to search through up to 65535 transparent latches per path until it reaches a nontransparent latch or an output.

-ignore_boundary_parasitics

Causes detailed parasitics annotated on the boundary nets to be ignored for model generation.

-extract_boundary_parasitics *file_format*

Causes the generated model to be in the form of a core cell and Verilog wrapper with boundary parasitics. Valid arguments are **spef** or **spef_unzipped**. In the absence of this option, the generated model is a library cell with the effects of boundary parasitics included in the timing arcs.

-test_design

Generates a test design containing an instance of the generated library cell. It creates a file called *name*_test that can be read into PrimeTime to test the model. This option can be used only when the **-extract_boundary_parasitics** option is not used.

-arc_types *arc_list*

Specifies a list of arc types to be included in the extracted model. If this option is not used, all arc types are included in the model. In the current release, the arc types that can be specified are **min** and **max**. If you specify **min**, only the timing arcs from the minimum-delay (shortest) paths are extracted.

If you specify **max**, only the timing arcs from the maximum-delay (longest) paths are extracted.

-ignore_ports ***ignore_port_list***

Specifies a list of ports that are ignored during model extraction and do not appear in the extracted model.

-extend_inputs ***extend_input_port_list***

For paths starting at the specified inputs, this option causes NanoTime to consider all endpoint sequential elements in the fanout of the input port. This increases the model's visibility into the design, at the cost of a larger model. Without this option, the model considers only the worst-case path to the single most constraining sequential element reachable from each input port.

-extend_outputs ***extend_output_port_list***

For paths ending at the specified outputs, this option causes NanoTime to consider all input-to-output timing arcs for model generation. Without this option, NanoTime considers only the input-to-output timing arcs for which the data arrival is later than for the worst clock-to-output timing arc. By default, the model generator only considers input-to-output timing arcs of transparent paths, where the arrival time is later than the worst clock-to-output arrival. Use the **-extend_outputs** option to specify the names of the output ports that require increased visibility. For those outputs, NanoTime considers all input-to-output timing arcs.

-debug ***debug_output_file_list***

Causes data files to be generated and written to the working directory for debugging purposes. Valid values of the *debug_output_file_list* are as follows: **lib**, **lib_only**, **paths**, **{lib paths}**, or **{lib_only paths}**. If you specify **lib**, NanoTime generates a .lib model file in Liberty format for the extracted model in addition to the .db file. If you specify **lib_only**, the .db file is not generated. If you specify **paths**, NanoTime generates a file that lists the specific worst case timing paths that are used to generate the timing model.

-non_transparent

Prevents the extraction of input-to-output transparent paths by treating output boundary latches (latches before output ports) as nontransparent. This results in a model with no tlatch constructs. Further, any setup timing checks derived from the output boundary latches are referenced to the opening clock edge.

-ignore_input_to_output_transparency

Ignores input-to-output transparent paths during model generation. This results in a model with no tlatch constructs. Further, any setup timing checks derived from the output boundary latches are referenced to the closing clock edge.

-combinational_input_to_output_transparency

Convert input-to-output transparent paths into combinational arcs during model generation. Does not apply to custom modeling. This results in a model with no tlatch constructs. Further, any setup timing checks derived from the output boundary latches are referenced to the closing clock edge.

-keep_paths_within ***time_value***

Keeps timing paths in the path database that have a path delay within the *time_value* of the worst path delay of each startpoint-endpoint pair. By default, the value of the **-keep_paths_within** option is set to 0.0, which keeps only the single worst path per startpoint-endpoint pair. To keep more paths, set the

-keep_paths_within value to a time value larger than 0.0.

-npaths ***number_of_paths***

Specifies the maximum number of paths to save in the path database for each startpoint-endpoint pair. The default is 1. Specify a larger number to save more paths, but use a value that is small enough to keep the runtime and memory usage reasonable.

-pbsa

Invokes path-based slack adjustment, which adjusts the calculated path delays based on the lengths of the path segments traced in the timing check. A NanoTime Ultra license is required to use this feature. For more information, see the NanoTime User Guide or the man page for the **report_pbsa_calculation** command.

-pocv

Invokes random on-chip variation path delay and slack adjustment, which adjusts the calculated path delays based on the timing variation from each path arc. A NanoTime Ultra license is required to use this feature. For more information, see the NanoTime User Guide.

-bus

Creates bus notation in the Verilog and .lib output. Applies to both the wrapper and test design.

-complete_bus

Completes the arcs for bus bits that do not have timing information using information from bits that do. This option requires a NanoTime Ultra license.

-mode ***mode_name***

Creates a moded model. Sets the name of the analysis mode for arcs in the model.

-use_find_path

Causes model extraction to add paths found by **set_find_path** commands.

-find_path_only

Causes model extraction to use only the paths found by **set_find_path** commands. Note that delays and transitions could be different for paths when they are created by the **set_find_path** command compared to when the same paths are created by the **trace_paths** command, due to dynamic effects such as signal integrity analysis.

-non_sequential_arcs

Causes model extraction to include nonsequential constraint arcs from data-to-data timing checks. Such timing checks include nonsequential setup and hold constraints as well as pulsewidth constraints. These pulsewidth constraints can be generated from explicit pulsewidth timing checks added with the **set_min_pulse_width** command or from setup and hold checks between a model pin and itself. Minimum clock period constraints are not supported.

-conform_arc_types

Causes model extraction to exclude any minimum delay arc that does not have a corresponding maximum delay arc.

-context_dependent

Causes model extraction to perform path tracing with the user-specified settings for input arrival times and transitions and output required times. This option is required for the effects of multicycle path commands to be included in the selection of critical paths for combinational and nontransparent delay arcs.

-extend_clock_domains

Causes model extraction to include setup and hold constraints based on timing checks to non-boundary clock domains, and also clock to output arcs from non-boundary clock domains. This option must be used with the **-context_dependent** option.

-enable_arc_segmentation

Causes model extraction to preserve half-unate arcs in the timing model. The default is to move arc boundaries and duplicate lookup tables to create full-unate model arcs. Ensure that the tool which consumes the extracted model can handle half-unate arcs before enabling this option. NanoTime can handle half-unate arcs.

-message_level ***level***

Specifies the level of detail in the messages produced during model extraction: *silent* (no messages), *normal* (the default), *verbose* (detailed messages), or *debug* (very detailed messages).

-hspice_timing

Causes model extraction to run HSPICE simulation to extract the timing model. Paths that are not simulated in HSPICE are marked with '~' in the .lib model file.

-when

Causes model extraction to insert a 'when' condition into arcs and other constructs in the model. The 'when' condition is a Boolean expression that is derived from the logic states of ports induced by the **set_case_analysis** command.

DESCRIPTION

The **extract_model** command generates a static timing model in .db format from the current design. The generated model has the same timing behavior as the original netlist. It can replace the original netlist in a higher-level timing analysis, allowing the analysis to be performed hierarchically. The model can be used in NanoTime and also can be used with other tools such as PrimeTime.

Before generating a timing model, be sure to apply the worst case clocking conditions, power supply levels, transistor technologies (with their associated operating conditions), and detailed parasitics. Specify the input delays, input transition times, output delays, and output loads that apply to the design from which you are extracting a model. Make sure that the design passes all timing checks by running the **trace_paths** command and the **report_constraint** or **report_paths** command.

Before you can perform model extraction, the design must be prepared in the same manner as for ordinary timing analysis, using the **link_design**, **match_topology**, **check_topology**, and **check_design** commands. During model extraction, NanoTime performs path tracing in a manner similar to the **trace_paths** command. However, path tracing for model extraction is more exhaustive than for normal timing analysis. Therefore, if you use the **trace_paths** command, you must reset the path database using

the **reset_design -paths** command before you can use the **extract_model** command.

The usability of the timing model is context independent for the following conditions (the model is accurate even when these condition change):

- data input arrival time
- data input transition time (slew)
- data output required time
- data output network load

The usability of the model is context dependent for the following conditions (the model is accurate only under these fixed conditions):

- clock frequency and duty cycle
- power rail voltages
- operating conditions

The **extract_model** command offers several options with respect to model type, the handling of boundary parasitics, the handling of transparent latches, and model size and speed versus accuracy and precision.

Model Types

The **extract_model** command supports the generation of the following types of timing models:

- library cell in .db and .lib format
- core cell, Verilog wrapper, and boundary parasitics
- custom model

To generate a library cell, omit the **-extract_boundary_parasitics** option. A library cell model replaces the entire design with a single, freestanding library cell that is easily used with other tools. The effects of boundary parasitics are included in the timing arcs of the model. The model consists of the library cell in .db (compiled) format and a command file in SDC (Synopsys Design Constraints) format that applies the applicable timing exceptions, such as false paths and multicycle paths, if any. The SDC file must be applied to the library cell when it is used at a higher level of the design hierarchy. To generate the same model in .lib format as well, use the **-debug {lib}** option.

To generate a core cell with a Verilog wrapper and boundary parasitics, use the **-extract_boundary_parasitics** option. This type of model maintains the boundary parasitics of the original netlist. Specify the format of the parasitic data: **spef** (gzip-compressed SPEF) or **spef_unzipped** (uncompressed SPEF).

In order to use the **-extract_boundary_parasitics** option, you must set the **timing_save_wire_delay** variable to **true** before the **check_design** command, and you must set the **rc_reduction_max_net_delta_delay** variable to zero or the **rc_reduction_exclude_boundary_nets** variable to **true** before using the **read_parasitics** command. You can use the **-ignore_ports** option at the same time if the ignored ports do not have parasitics.

To build a custom timing model in your own proprietary format, first use the **extract_model**

command, then get the model and timing arc information with the following commands:

```
get_model_clock_domains
get_capture_window_edges
get_launch_window_edges
get_model_delay_arcs
get_attribute
```

Creating a custom model requires Tcl programming. You are responsible for ensuring compatibility of the custom model with any subsequent analysis tool.

Use the **-name** option to specify the base name for generating the model files. If you only want to generate a custom model, no model files are necessary and you can omit the **-name** option.

CCS Modeling

By default, the **extract_model** command produces a library cell using nonlinear delay model (NLDM) modeling only. To produce a model using the more advanced composite current source (CCS) timing modeling, use the **-library_elements** option and specify **{nldm ccs_timing}** as the list of element types, which causes both NLDM and CCS type modeling to be produced. The default is **nldm** alone, which produces NLDM type modeling only.

To produce a CCS noise model, use the **-library_elements** option and specify **{nldm ccs_timing ccs_noise}** as the list of element types. CCS noise data is produced along with NLDM and CCS timing data.

If any CCS modeling is specified, the **extract_model** command produces the model in .db and optionally in .lib formats. The generation of .db is possible only if the .lib has no errors. The **extract_model** command supports CCS timing and CCS noise (but not CCS power). The generated CCS model is valid only for the single operating condition in effect at the time of model extraction. For modeling of different worst case operating conditions, it is necessary to generate a separate model for each condition.

The following variables control the output waveform segmentation:

```
model_ccs_measured_delay_tolerance
model_ccs_waveform_segment_tolerance
model_ccs_characterization_driver_type
```

The **model_ccs_measured_delay_tolerance** 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. The default is 0.02.

The **model_ccs_waveform_segment_tolerance** 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 are used in waveform tables. The default is 0.005.

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

Latch Transparency Depth

By default, the extracted model preserves the timing behavior of the worst-case paths, including any number of successive transparent latches per path.

If you know the maximum number of successive transparent latches in any given path within the whole

current design, you can specify that number with the **-latch_level** option. Doing so prevents the model generator from spending time trying to trace through more transparent latches than necessary. In addition, it helps ensure accurate identification of longer paths where the circuit structure might resemble a transparent latch.

Worst-Case Paths Used in the Model

The extracted timing model preserves the timing behavior of the worst-case paths in the design. You can optionally increase the number of paths considered for model generation by using the **-extend_inputs** and **-extend_outputs** options.

By default, the extracted model preserves the timing behavior of the single worst-case path from each input to the single most constraining sequential element that captures data in the fanout of the input. You can optionally have NanoTime consider all sequential elements in the fanout of an input, thereby increasing the model's visibility into the design. In that case, if you change the external conditions to correct the worst violation, the same model can still detect other possible violations, such as in the path to the second-worst or third-worst sequential element. Use the **-extend_inputs** option to specify the names of the input ports that require increased visibility. Note that this can result in a much larger model.

Each output of the design can have both clock-to-output and input-to-output timing arcs. An output is said to be transparent if there is at least one input-to-output path in which the data arrives later than the worst clock-to-output arrival. By default, the model generator only considers input-to-output timing arcs of transparent paths, where the arrival time is later than the worst clock-to-output arrival. Use the **-extend_outputs** option to specify the names of the output ports that require increased visibility. For those outputs, NanoTime considers all input-to-output timing arcs.

To find out which paths are the worst-case paths used to generate the extracted model, use the **-debug {paths lib}** option of the **extract_model** command. This option causes the model generator to write out a .lib source file and another file that describes worst-case paths chosen to generate the timing model. The .lib file has comment lines that show which timing arcs were generated from which paths. The presence of timing errors in the design can cause certain paths through transparent devices to become infeasible. Such paths are marked with the "*" character in the .lib file. The worst-case path file appears in the current directory and is named *model_name*.paths.

Transition and Load Index Values

The generated timing model adjusts its behavior according to the input transition times and output loading conditions that occur where the model is being used. At model creation time, NanoTime analyzes the circuit behavior under specific conditions and generates lookup tables for calculating the changes in timing behavior. The specific transition time and load values used in the lookup tables are called index values. When the model is being used, the analysis tool uses interpolation for conditions between the index values, and extrapolation for conditions beyond the index values, to calculate the timing parameters.

Before you use the **extract_model** command, you can specify the range and spacing of lookup table entries in the extracted model by using the following commands:

```
set_model_input_transition_indexes
set_model_load_indexes
```

In the absence of these commands, the model generator gets the lookup table index values from the following variables:

```
model_default_input_transition_indexes
```

```
model_default_load_indexes
```

If the **-library_elements** option is set to **{nldm ccs_timing}**, and the design for which you are generating an extracted model contains library cells, and a CCS-based library cell drives an output port of the design, then NanoTime uses the load index values of the existing CCS driver model. Under these conditions, NanoTime ignores the load index values specified by the **set_model_load_indexes** command and the **model_default_load_indexes** variable.

NanoTime uses the input transition index values specified by the **set_model_input_transition_indexes** command or the **model_default_input_transition_indexes** variable, regardless of any library cells used in the design.

Bus Notation

If you use the **-bus** option and preserve the boundary parasitics, the **extract_model** command uses bus notation in the generated output. For example:

```
nt_shell> extract_model -bus ... \
          -extract_boundary_parasitics spef
```

NanoTime generates a Verilog wrapper file with syntax similar to the following:

```
module ALU (
        or ,
   ...
        ABUS ,
        BBUS ,
        SBUS ,
        SELS ,
        HOLDA ,
        HOLDB ,
        HOLDS
);

input
        or ,
   ...
        [0:63] ABUS ,
        [0:63] BBUS ,
        SELS ,
        HOLDA ,
        HOLDB ,
        HOLDS ;
```

The generated .lib file has syntax similar to the following:

```
type ( BUS64_type0 ) {
        base_type : array ;
        data_type : bit ;
        bit_width : 64 ;
        bit_from : 0 ;
        bit_to : 63 ;
} /* end of type */

bus ( ABUS ) {

      bus_type  : BUS64_type0 ;
      direction : input ;
```

```
        ...

        pin(ABUS[23]) {
                direction : input ;
                capacitance : 0.028058 ;
                rise_capacitance_range (0.028058, 0.028058);
                fall_capacitance_range (0.028058, 0.028058);
        } /* end of pin ABUS[23] */

} /* end of bus ABUS */
```

Path-Based Slack Adjustment (PBSA)

The **-pbsa** option invokes path-based slack adjustment, which adjusts the calculated path delays according to the lengths of the path segments traced in the timing check. A timing analysis using path-based slack adjustment results in less pessimism than setting global clock uncertainty values with the **set_clock_uncertainty** command, but requires more effort and runtime. For details, see the NanoTime User Guide or the man page for the **report_pbsa_calculation** command.

Parametric On-Chip Variation (POCV)

The **-pocv** option invokes parametric on-chip variation analysis, which accounts for local fluctuations in device behavior using statistical modeling techniques. You must define the variation analysis conditions by using commands such as the **set_variation_parameters** command and variables such as the **timing_pocv_sigma** variables.

A NanoTime Ultra license is required to use POCV. For more information, see the NanoTime User Guide.

Model Testing

To test a new timing model, use the **-test_design** option. The **-test_design** option generates a test design containing one instance of a generated library cell named "core," with ports connected to the cell inputs and outputs. The name assigned to the test design is *name*_test. You can use the PrimeTime tool to perform a timing analysis of this test design to confirm that the library cell has the same characteristics as the original netlist.

Restricting Timing Arc Types

To restrict the arc types produced in the extracted model, use the **-arc_types** option and specify the types of arcs you want to extract. The arc types that can be specified are **min** and **max**. If you specify **min**, only the timing arcs from the minimum-delay (shortest) paths are extracted. If you specify **max**, only the timing arcs from the maximum-delay (longest) paths are extracted. In the absence of the **-arc_types** option, all arc types are extracted.

The set_find_path Command

By default, the **extract_model** command cannot be used after **set_find_path** commands have been issued because model generation uses its own complete set of path tracing. However, if you have a specific path in mind to use for creating a constraint arc or delay arc (or both) for a model, and you would rather have that path used than the worst-case path found by normal path tracing, you can do so. Execute one or more **set_find_path** commands, then execute the **extract_model** command with the **-use_find_path** option. In that case, the **extract_model** command performs the **set_find_path** command searches after all normal model tracing has been completed. Any path found by a search using the **set_find_path** command replaces a path saved earlier, even it is subcritical (not the worst-case path).

Dynamic Clock Simulation

When the **extract_model** command is used in the context of dynamic clock simulation, the arcs in the model that come from dynamic clock simulation are given constant delay values. Such arcs do not have any sensitivity to changes in input slope or output load. (Clock arcs typically end on internal pins where the output load is constant anyway.)

COMMAND PHASING

The **extract_model** command can only be executed in the "design-checked" state (after the **check_design** command). After successful execution, the state is changed to "paths-traced."

EXAMPLES

The following command generates a timing model in NLDM format for the current design as a .db library cell. The model is generated for the current operating conditions set on the design. The model files are contained in a directory called DMA in the current working directory.

```
nt_shell> extract_model -name DMA
```

The following command generates a timing model for the current design, producing a model in .db format containing both NLDM and CCS timing modeling data.

```
nt_shell> extract_model -name DMA  \
          -library_elements {nldm ccs_timing }
```

SEE ALSO

```
create_clock(2)
get_capture_window_edges(2)
get_launch_window_edges(2)
get_model_delay_arcs(2)
get_model_clock_domains(2)
report_pbsa_calculation(2)
set_variation_parameters(2)
timing_pocv_sigma(3)
set_find_path(2)
model_ccs_characterization_driver_type(3)
model_ccs_measured_delay_tolerance(3)
model_ccs_waveform_segment_tolerance(3)
model_default_input_transition_indexes(3)
model_default_load_indexes(3)
rc_reduction_max_net_delta_delay(3)
timing_save_wire_delay(3)
trace_paths(2)
```

filter_collection

Filters an existing collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection filter_collection
   [-regexp]
   [-nocase]
   base_collection
   expression
```

Data Types

base_collection	collection
expression	string

ARGUMENTS

-regexp

Specifies that the == and != filter operators use real regular expressions. By default, the == and != filter operators use simple wildcard pattern matching with the * and ? wildcards.

-nocase

Makes the pattern match case-insensitive. When you specify this option, you must also specify the **-regexp** option.

base_collection

Specifies the base collection to be filtered. This collection is copied to the result collection. Objects are removed from the result collection if they are evaluated as **false** by the conditional *expression* value.

expression

Specifies an expression with which to filter *base_collection*.

DESCRIPTION

The **filter_collection** command filters an existing collection, resulting in a new collection. The base collection remains unchanged.

In many cases, commands that create collections support a **-filter** option that filters as part of the collection process, rather than after the collection has been made. That type of filtering is almost always more efficient than using the **filter_collection** command after a collection has been formed. The **filter_collection** command is most useful if you plan to filter the same large collection many times using different criteria.

The **filter_collection** command and the **-filter** option of collection-creating commands operate on object attributes. To get a complete listing of attributes that can be filtered, use the following command:

```
nt_shell>  list_attributes -application -class object_type
```

where *object_type* is the object type, such as design, port, cell, or net.

The **filter_collection** command results in either a new collection or an empty string. A resulting new collection contains a subset of the objects in the input *base_collection*. A resulting empty string (the empty collection) indicates that the *expression* filtered out all elements of the input *base_collection*.

The basic form of the conditional expression is a series of relations joined together with AND and OR operators. Parentheses () are also supported. The basic relation contrasts an attribute name with a value through a relational operator. For example:

```
is_hierarchical == true and area <= 6
```

The relational operators are

```
==   Equal
!=   Not equal
>    Greater than
<    Less than
>=   Greater than or equal to
<=   Less than or equal to
=~   Matches pattern
!~   Does not match pattern
```

The basic relational rules are

- String attributes can be compared with any operator.
- Numeric attributes cannot be compared with pattern match operators.
- Boolean attributes can be compared only with == and !=. The value can be only **true** or **false**.

Existence relations determine whether an attribute is defined for the object. For example:

```
sense == setup_clk_rise and defined(sdf_cond)
```

The existence operators are **defined** and **undefined**. These operators apply to any attribute if it is valid for the object class.

The **-regexp** option performs regular expression matching in the same way as the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the filter expression. Using rigid quoting with curly braces around regular expressions is recommended.

Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expressions as needed. You can make the regular expression search case-insensitive by using the **-nocase** option.

COMMAND PHASING

The **filter_collection** command can be executed at any time.

EXAMPLES

The following example creates a collection of only hierarchical cells.

```
nt_shell> set a [filter_collection [get_cells *] \
"is_hierarchical == true"]
{"xck1", "xck2", "xck3"}
```

SEE ALSO

collections(2)

foreach_in_collection

Iterates over the elements of a collection.

SYNTAX

```
status foreach_in_collection
   itr_var
   collections
   body
```

Data Types

```
   itr_var          string
   collections      list
   body             string
```

ARGUMENTS

itr_var

Specifies the name of the iterator variable.

collections

Specifies a list of collections over which to iterate.

body

Specifies the commands to execute during each iteration.

DESCRIPTION

The **foreach_in_collection** command iterates over each element in a collection. You cannot use the Tcl-supplied **foreach** command to iterate over collections, because the **foreach** command requires a list, and a collection is not a list. Using the **foreach** command on a collection deletes the collection.

The arguments for the **foreach_in_collection** command are as follows: an iterator variable, the collections over which to iterate, and the commands to apply during each iteration. All arguments are required. Note that the **foreach_in_collection** command supports only one iterator variable; it does not allow a list.

During each iteration, the iteration variable *itr_var* is set to a collection of exactly one object, stepping through each object in the collection. Any command that accepts a collection as an argument accepts the iteration variable *itr_var*.

You can nest the **foreach_in_collection** command within other control structures, including other **foreach_in_collection** commands.

If commands in the body of the iteration modify the netlist, it is possible that all or part of the collection involved in the iteration will be deleted. The **foreach_in_collection** command is safe for such operations. If a command in the body causes the collection to be removed, at the next iteration the loop ends with a message indicating that the iteration ended prematurely.

An alternative to iteration over a collection is to use filtering to create a collection that includes only the desired elements, then apply one or more commands to that collection. If the order of operations does not matter, the following two examples are equivalent. The first is an example without iterators.

```
set s [get_cells {U1/*}]
command1 $s
command2 $s
unset s
```

The second example uses the **foreach_in_collection** command.

```
foreach_in_collection itr [get_cells {U1/*}] {
  command1 $itr
  command2 $itr
}
```

For collections with large numbers of objects, the non-iterator version is more efficient, though both produce the same results if the commands are order-independent.

COMMAND PHASING

The **foreach_in_collection** command can be executed at any time.

EXAMPLES

The following example sets all latches in the design to non-transparent.

```
nt_shell> set latches [get_topology -structure_type latch]
nt_shell> foreach_in_collection itr $latches {
             set_non_transparent $itr
             }
```

SEE ALSO

```
collections(2)
filter_collection(2)
query_objects(2)
```

foreach_match

Applies a command to all occurrences of a subcircuit or pattern in the design.

SYNTAX

```
status foreach_match
   [-pattern]
   [-silent]
   -command command
   name_list
```

Data Types

```
   command          string
   name_list        list
```

ARGUMENTS

-pattern

Matches subcircuits by pattern rather than by name.

-silent

Suppresses error messages during execution of the command.

-command *command*

Specifies the command to apply to the matched instance.

name_list

Lists the names of the subcircuits or patterns to match.

DESCRIPTION

The **foreach_match** command identifies all instances of a specified pattern or named subcircuit and applies a specified command to each matching instance. The matching is pattern-based if you use the **-pattern** option, or name-based otherwise.

If you use the **-pattern** option, the pattern must be read into NanoTime beforehand with the **read_pattern** command. To find out the names of subcircuits that have been read in as patterns, use the **list_patterns** command. For more information about matching based on patterns, see the man page for the **read_pattern** command.

COMMAND PHASING

The **foreach_match** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command applies the **set_transistor_direction** command to all instances of the subcircuit named MUX, thereby specifying the signal direction as source-to-drain for transistor MN1 in MUX.

```
nt_shell> foreach_match MUX \
          -command { set_transistor_direction \
          -transistor s2d MN1 }
```

The following example reads the pattern file named pattern.sp and lists the names of the subcircuits that have been read in as patterns. Then, for each matching occurrence of the pattern in the current design, it applies an **erase_turnoff** command at output net z.

```
nt_shell> read_pattern pattern.sp
Compiling "/node1/user1/designs/pattern.sp"
1
nt_shell> list_patterns
Pattern Registry

Pattern Name
-----------------------------------------
pre_an2k
1
nt_shell> foreach_match -pattern pre_an2k \
          -command {erase_turnoff -output_net z}
1
1
1
```

SEE ALSO

```
list_patterns(2)
read_pattern(2)
remove_pattern(2)
```

get_app_var

Gets the value of an application variable.

SYNTAX

```
string get_app_var
   [-default | -details | -list]
   [-only_changed_vars]
   var
```

Data Types

```
   var     string
```

ARGUMENTS

-default

Gets the default value.

-details

Gets additional variable information.

-list

Returns a list of variables matching the pattern. When this option is used, then the *var* argument is interpreted as a pattern instead of a variable name.

-only_changed_vars

Returns only the variables matching the pattern that are not set to their default values, when specified with **-list**.

var

Specifies the application variable to get.

DESCRIPTION

The **get_app_var** command returns the value of an application variable.

There are four legal forms for this command:

- `get_app_var <var>`
  `Returns the current value of the variable.`

- `get_app_var <var> -default`
  `Returns the default value of the variable.`

- `get_app_var <var> -details`

Returns more detailed information about the variable. See below for details.

- get_app_var -list [-only_changed_vars] <*pattern*>

Returns a list of variables matching the pattern. If *-only_changed_vars* is specified, then only variables that are changed from their default values are returned.

In all cases, if the specified variable is not an application variable, then a Tcl error is returned, unless the application variable **sh_allow_tcl_with_set_app_var** is set to true. See the **sh_allow_tcl_with_set_app_var** man page for details.

When **-details** is specified, the return value is a Tcl list that is suitable as input to the Tcl **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key, and each even-numbered element in the list is the value of the previous key.

The supported keys are as follows:

name

This key contains the name of the variable. This key is always present.

value

This key contains the current value of the variable. This key is always present.

default

This key contains the default value of the variable. This key is always present.

help

This key contains the help string for the variable. This key is always present, but sometimes the value is empty.

type

This key contains the type of the application variable. Legal values of for this key are: string, bool, int, real. This key is always present.

constraint

This key describes additional constraints placed on this variable. Legal values for this key are: none, list, range. This key is always present.

min

This key contains the min value of the application variable. This key is present if the constraint is range. The value of this key may be the empty string, in which case the variable only has a max value constraint.

max

This key contains the max value of the application variable. This key is present if the constraint is "range". The value of this key may be the empty string, in which case the variable only has a min value constraint.

list

This key contains the list of legal values for the application variable. This key is present if the constraint is "list".

EXAMPLES

The following are examples of the **get_app_var** command:

```
prompt> get_app_var sh_enable_page_mode
1

prompt> get_app_var sh_enable_page_mode -default
false

foreach {key val} [get_app_var sh_enable_page_mode -details] {   echo "$key: $val"
}
=>  name: sh_enable_page_mode
    value: 1
    default: false
    help: Displays long reports one page at a time
    type: bool
    constraint: none

prompt> get_app_var -list sh_*message
sh_new_variable_message
```

SEE ALSO

```
report_app_var(2)
set_app_var(2)
write_app_var(2)
```

get_attribute

Retrieves the value of an attribute on an object.

SYNTAX

```
status get_attribute
   [-class class_name]
   [-quiet]
   [-value_list]
   object_spec
   attr_name
```

Data Types

```
   class_name          string
   object_spec         string
   attr_name           string
```

ARGUMENTS

-class *class_name*

The class name of *object_spec*, if *object_spec* is a name. You must use this option if *object_spec* is a name.

-quiet

Prevents error and warning messages from being reported.

-value_list

Forces single-object return as list.

object_spec

A single object from which to get the attribute value. The *object_spec* must be either a collection of exactly one object, or a name which is combined with the **-class** option to find the object.

attr_name

The name of the attribute.

DESCRIPTION

The **get_attribute** command retrieves the value of an attribute on an object. The return value is a string.

Valid values for the **-class** option are:

```
capture_window_edge
cell
clock
design
launch_window_edge
lib
lib_cell
lib_pin
lib_topology
model_clock_domain
model_delay_arc
net
parasitic_device
path_arc
pin
port
simulation
timing_check
timing_path
timing_point
topology
```

To get a list of available attributes, use the **list_attributes -application** command.

To report the values of all attributes associated with an object, use the **report_attribute** command.

COMMAND PHASING

The **get_attribute** command is not phase-restricted.

EXAMPLES

The following command gets an attribute for a specific object:

```
nt_shell> get_attribute [get_nets Sn1] total_capacitance_fall_max
0.039744
```

The following command gets an attribute for cell instance Xxor7:

```
nt_shell> get_attribute -class cell [get_cells Xxor7] is_transistor
false
```

SEE ALSO

```
list_attributes(2)
report_attribute(2)
```

get_capture_window_edges

Returns a collection of capture window edges for a specified input port, obtained from a path database created by the **extract_model** command.

SYNTAX

```
collection get_capture_window_edges
   -start | -end | -transparent
   -input_rising input_port | -input_falling input_port
   -clock_domain model_clock_domain
   [-output output_port]
   [-worst i]
```

Data Types

```
   input_port              string
   model_clock_domain      string
   output_port             string
   i                       int
```

ARGUMENTS

-start

Gets the capture_window_edge object for the start of the capture window.

-end

Gets the capture_window_edge object for the end of the capture window.

-transparent

Gets the capture_window_edge object for the start of the capture window, for the case of a transparent path from input to output. Must be used together with the **-output** option.

-input_rising *input_port*

Gets the capture_window_edge object for logic 1 input data.

-input_falling *input_port*

Gets the capture_window_edge object for logic 0 input data.

-clock_domain ***model_clock_domain***

Specifies the clock domain of the input port.

-output ***output_port***

Gets the capture_window_edge object for the start of the capture window, for a transparent path from the specified input to the specified output (used only with the **-transparent** option).

-worst ***i***

Gets the capture_window_edge object for the starting event of the *i*-th most constraining sequential element downstream from the input port. The default is 1. The valid range is from 1 to *n*, where *n* is the number specified in the **-extend_inputs** option of the **extract_model** command. The **-worst** option can be used only with the **-start** option.

DESCRIPTION

The **extract_model** command generates a path database from the current design. You can use information from this database to generate a timing model in your own proprietary format. The following commands are available for getting the path database information:

- get_model_clock_domains
- get_capture_window_edges
- get_launch_window_edges
- get_model_delay_arcs
- get_attribute

The **get_capture_window_edges** command gets the starting or ending capture window edge object associated with either a rising or falling transition at a specified input, for a specified capture clock domain.

The capture window is the time span during which the input data must be valid and stable to ensure reliable capture of the data at a clocked storage element downstream from the input. The start of the capture window is the latest time at which the specified input transition can occur, when the input data is first required to be valid. The end of the capture window is the earliest time that the input data is allowed to change and become invalid.

The command returns a collection of capture_window_edge objects. If no appropriate capture_window_edge object can be found, a zero-length collection is returned.

For a path that ends at a sequential element inside the timing model, you specify the window edge (start or end) input port, input edge type (rising or falling) and capture clock domain.

For a transparent path from input to output, you must use the **-transparent** and **-output** options, and the command returns the capture_window_edge object for the start of the capture window. You specify the output port, input port, input edge type (rising or falling) and capture clock domain.

If the **-extend_inputs** option of the **extract_model** command was used to generate the path database, you can optionally use the **-worst** option of the **get_capture_window_edges** command to get the capture_window_edge object for a path to one of the alternative sequential elements in the database. For example, use **-worst 2** to get information on the worst-case path to the second-most constraining

sequential element in the model.

A capture_window_edge object in the collection has the following attributes:

- input_arrival_time
- model_delay_arc
- output_clock_domain (for a transparent path only)

After using the **get_capture_window_edges** command, you can iterate through the collection with the **foreach_in_collection** command and get the capture_window_edge attributes by using the **get_attribute** command.

COMMAND PHASING

The **get_capture_window_edges** command can only be executed after path tracing is complete.

EXAMPLES

The following command gets the capture_window_edge object for the starting edge of the stable-low capture window, for the transparent path from input data[7] to output result[7]. The command sets the variable "edges1" to the resulting collection.

```
nt_shell> set edges1 [get_capture_window_edges \
          -transparent -input_falling "data[7]" \
          -clock_domain $clock_domain \
          -output "result[7]"]
```

SEE ALSO

collections(2)
extract_model(2)
get_attribute(2)
get_model_clock_domains(2)
get_launch_window_edges(2)
get_model_delay_arcs(2)

get_cells

Creates a collection of cells from the current design relative to the current instance. You can assign these cells to a variable or pass them into another command.

SYNTAX

```
collection get_cells
   [-hierarchical]
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns | -of_objects objects
```

Data Types

```
   expression       string
   patterns         list
   objects          list
```

ARGUMENTS

-hierarchical

Searches for cells level by level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a cell block1/adder, a hierarchical search finds it using "adder".

-filter *expression*

Filters the collection with *expression*. For any cells that match *patterns* (or *objects*) the expression is evaluated based on the cell's attributes. If the expression evaluates to **true**, the cell is included in the result.

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching, for searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** options are mutually exclusive.

-of_objects ***objects***

Creates a collection of cells connected to the specified objects. In this case, each object is either a named pin or a pin collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both. In addition, you cannot use the **-hierarchical** option with the **-of_objects** option.

patterns

Matches cell names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type "cell."

DESCRIPTION

The **get_cells** command creates a collection of cells in the current design, relative to the current instance, that match certain criteria. In NanoTime, cells are used to represent the netlist hierarchy as well as primitive elements such as transistors, resistors, capacitors, and model instances. The command returns a collection if any cells match *patterns* or *objects* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

If *patterns* (or *objects*) fail to match any objects and the current design is not linked, the design is automatically linked.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_cells** command at the command prompt, or you can nest it as an argument for another command (such as the **query_objects** command). In addition, you can assign the **get_cells** command result to a variable.

When issued from the command prompt, the **get_cells** command behaves as though a **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_cells** command provides a fast, simple way to display cells in a

collection. However, if you want the flexibility provided by the **query_objects** command options (for example, if you want to display the object class), use the **get_cells** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_cells** command is not phase-restricted.

EXAMPLES

The following command queries the cells that contain the string "ck" and reference an "inv" library cell. Although the output looks like a list, it is not. The output is just a display.

```
nt_shell> get_cells "*ck*" -filter "ref_name == inv"
{"xck1", "xck2", "xck3"}
```

The following example shows that, given a collection of pins, you can query the cells connected to those pins.

```
nt_shell> set pinsel [get_pins *ck*.a]
{"xck1.a", "xck2.a", "xck3.a"}
nt_shell> query_objects [get_cells -of_objects $pinsel]
{"xck1", "xck2", "xck3"}
```

The following example gets all transistor cells in the current instance.

```
nt_shell> get_cells * -filter "is_transistor == true"
{"xl1.mn1"}
```

The following example sets the width of transistor cells mn1 and mp1.

```
nt_shell> set_transistor_parameter -width 0.25 \
                  [get_cells {mn1 mp1}]
1
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_pins(2)
link_design(2)
query_objects(2)
collection_result_display_limit(3)
```

get_clocks

Creates a collection of clocks from the current design. You can assign these clocks to a variable or pass them into another command.

SYNTAX

```
collection get_clocks
   [-quiet]
   [-regexp]
   [-nocase]
   [-filter expression]
   patterns
```

Data Types

```
   expression      string
   patterns        list
```

ARGUMENTS

-quiet

Suppresses warning and error messages if no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-filter *expression*

Filters the collection with *expression*. For any clocks that match *patterns*, the expression is evaluated based on the clock attributes. If the expression evaluates to **true**, the clock is included in the result.

patterns

Matches clock names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_clocks** command creates a collection of clocks in the current design that match certain criteria. The command returns a collection if any clocks match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_clocks** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_clocks** command result to a variable.

When issued from the command prompt, the **get_clocks** command behaves as though a **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_clocks** command provides a fast, simple way to display clocks in a collection. However, if you want the flexibility provided by the **query_objects** command options (for example, if you want to display the object class), use the **get_clocks** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page. In addition, see the man page for **all_clocks**, which also creates a collection of clocks.

COMMAND PHASING

The **get_clocks** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example applies the **set_clock_uncertainty** command to all clocks in the design that match the "PHI*" string.

```
nt_shell> set_clock_uncertainty 0 [get_clocks "PHI*"]
```

The following example sets the variable clock_list to a collection of clocks that have a period of 15.

```
nt_shell> set clock_list [get_clocks * -filter "period==15"]
```

SEE ALSO

```
all_clocks(2)
collections(2)
create_clock(2)
create_generated_clock(2)
query_objects(2)
report_clock(2)
collection_result_display_limit(3)
```

get_command_option_values

Queries current or default option values.

SYNTAX

```
get_command_option_values
   [-default | -current]

   -command command_name
```

Data Types

```
   command_name      string
```

ARGUMENTS

-default

Gets the default option values, if available.

-current

Gets the current option values, if available.

-command *command_name*

Gets the option values for this command.

DESCRIPTION

This command attempts to query a default or current value for each option (of the command) that has default and/or current-value-tracking enabled. Details of how the option value is queried depend on whether one of the **-current** or **-default** options is specified (see below).

A "Tcl array set compatible" (possibly empty) list of option names and values is returned as the Tcl result. The even-numbered entries in the list are the names of options that were enabled for default-value-tracking or current-value-tracking and had at least one of these values set to a not-undefined value). Each odd-numbered entry in the list is the default or current value of the option name preceding it in the list.

Any options that were not enabled for either default-value-tracking nor current-value-tracking are omitted from the output list. Similarly, options that were enabled for default-value-tracking or current-value-tracking, but for which no (not-undefined) default or current value is set, are omitted from the result list.

If neither **-current** nor **-default** is specified, then for each command option that has either default-value-tracking or current-value-tracking (or both) enabled, the value returned is as follows:

- The current value is returned if current-value-tracking is enabled and a (not-undefined) current value has been set;
- Otherwise the default value is returned if default-value-tracking is enabled and a (not-undefined) default value has been set;
- Otherwise the name and value pair for the option is not included in the result list.

If **-current** is specified, the value returned for an option is the current value if current-value-tracking is enabled, and a (not-undefined) current value has been set; otherwise the name and value pair for the option is omitted from the result list.

If **-default** is specified, the value returned for an option is the default value if default-value-tracking is enabled, and a (not-undefined) default value has been set; otherwise the name and value pair for the option is omitted from the result list.

The result list from **get_command_option_values** includes option values of both dash options and positional options (assuming that both kinds of options of a command have been enabled for value-tracking).

The command issues a Tcl error in a variety of situations, such as if an invalid command name was passed in with **-command**.

EXAMPLES

The following example shows the use of **get_command_option_values**:

```
prompt> test -opt1 10 -opt2 20
1

prompt> get_command_option_values -command test
-bar1 10 -bar2 20
```

SEE ALSO

```
preview(2)
set_command_option_value(2)
```

get_defined_commands

Get information on defined commands and groups.

SYNTAX

```
string get_defined_commands [-details]
   [-groups]
   [pattern]

string pattern
```

ARGUMENTS

-details

Get detailed information on specific command or group.

-groups

Search groups rather than commands

pattern

Return commands or groups matching pattern. The default value of this argument is "*".

DESCRIPTION

The **get_defined_commands** gets information about defined commands and command groups. By default the command returns a list of commands that match the specified pattern.

When **-details** is specified, the return value is a list that is suitable as input to the **array set** command. The returned value is a list with an even number of arguments. Each odd-numbered element in the list is a key name, and each even-numbered element in the list os the value of the previous key. The **-details** option is only legal if the pattern matches exactly one command or group.

When **-group** is specified with **-details**, the supported keys are as follows:

name

This key contains the name of the group.

info

This key contains the short help for the group.

commands

This key contains the commands in the group

When **-details** is used with a command, the supported keys are as follows:

name

This key contains the name of the command.

info

This key contains the short help for the command.

groups

This key contains the group names that this command belongs to.

options

This key contains the options defined for the command. The value is a list.

return

This key contains the return type for the command.

Each element in the *options* list also follows the key value pattern. The set of available keys for options are as follows:

name

This key contains the name of the option.

info

This key contains the short help for the option.

value_info

This key contains the short help string for the value

type

The type of the option.

required

The value will be 0 or 1 depending if the option is optional or required.

is_list

Will be 1 if the option requires a list.

list_length

If a list contains the list length constraint. One of any, even, odd, non_empty or a number of elements.

allowed_values

The allowed values if the option has specified allowed values.

min_value

The minimum allowed value if the option has specified one.

max_value

The maximum allowed value if the option has specified one.

EXAMPLES

```
prompt> get_defined_commands *collection
add_to_collection append_to_collection copy_collection filter_collection
foreach_in_collection index_collection sort_collection
prompt> get_defined_commands -details sort_collection
name sort_collection info {Create a sorted copy of the collection}
groups {} options {{name -descending info {Sort in descending order}
value_info {} type boolean required 1 is_list 0} {name -dictionary info
{Sort strings dictionary order.} value_info {} type boolean required 1
is_list 0} {name collection info {Collection to sort} value_info
collection type string required 0 is_list 0} {name criteria info {Sort
criteria - list of attributes} value_info criteria type list required 0
is_list 1 list_length non_empty}}
```

SEE ALSO

```
help(2)
man(2)
```

get_designs

Creates a collection of one or more designs loaded into NanoTime. You can assign these designs to a variable or pass them into another command.

SYNTAX

```
collection get_designs
   [-hierarchical]
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns
```

Data Types

expression	string
patterns	list

ARGUMENTS

-hierarchical

Searches for designs inferred by the design hierarchy relative to the current instance. The full name of the object at a particular level must match the patterns. The use of this option does not force automatic linking.

-filter *expression*

Filters the collection with *expression*. For any designs that match *patterns*, the expression is evaluated based on the design attributes. If the expression evaluates to **true**, the design is included in the result.

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set if real regular expressions rather than simple wildcard patterns.

Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters.

patterns

Matches design names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type "design."

DESCRIPTION

The **get_designs** command creates a collection of designs from those currently loaded into NanoTime that match certain criteria. The command returns a collection if any designs match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using **-regexp**, take care in the way you quote the *patterns* and filter *expression*. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search simply by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_designs** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the **get_designs** command result to a variable.

When issued from the command prompt, the **get_designs** command behaves as though a **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_designs** command provides a fast, simple way to display clocks in a collection. However, if you want the flexibility provided by the **query_objects** command options (for example, if you want to display the object class), use the **get_designs** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_designs** command is not phase-restricted.

EXAMPLES

The following example queries the designs that begin with the letter t. Although the output looks like a list, it is just a display. A complete listing of designs is available using the **list_designs** command.

```
nt_shell> get_designs t*
{"top"}
```

SEE ALSO

collections(2)
filter_collection(2)
list_designs(2)
query_objects(2)
collection_result_display_limit(3)

get_generated_clocks

Creates a collection of generated clocks.

SYNTAX

```
collection get_generated_clocks
   [-quiet]
   [-regexp]
   [-nocase]
   [-filter expression]
   patterns
```

Data Types

```
   expression       string
   patterns         list
```

ARGUMENTS

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-filter *expression*

Filters the collection with *expression*. For any generated clocks that match *patterns*, the expression is evaluated based on the generated clock attributes. If the expression evaluates to **true**, the generated clock is included in the result.

patterns

Matches generated clock names against patterns. Patterns can include the wildcard characters "*" and "?".

DESCRIPTION

The **get_generated_clocks** command creates a collection of generated clocks from the current design that match certain criteria. The command returns a collection if any generated clocks match the *patterns* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

You can use the **get_generated_clocks** command at the command prompt, or you can nest it as an argument to another command (such as the **query_objects** command). In addition, you can assign the **get_generated_clocks** command result to a variable.

When issued from the command prompt, the **get_generated_clocks** command behaves as though a **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_generated_clocks** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command options (for example, if you want to display the object class), use the **get_generated_clocks** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_generated_clocks** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command applies the **set_clock_latency** command on all generated clocks in the design matching the "GEN*" string.

```
nt_shell> set_clock_latency 2.0 [get_generated_clocks "GEN*"]
```

The following command removes all the generated clocks in the design matching the "GEN1*" string.

```
nt_shell> remove_generated_clock [get_generated_clocks "GEN1*"]
```

SEE ALSO

```
collections(2)
create_generated_clock(2)
remove_generated_clock(2)
report_clock(2)
query_objects(2)
collection_result_display_limit(3)
```

get_launch_window_edges

Returns a collection of launch window edges for a specified output port, obtained from a path database created by the **extract_model** command.

SYNTAX

```
collection get_launch_window_edges
   -start | -end | -transparent
   -output_rising output_port | -output_falling output_port
   -clock_domain model_clock_domain
   [-input input_port]
```

Data Types

```
   output_port             string
   model_clock_domain      string
   input_port              string
```

ARGUMENTS

-start

Gets the launch_window_edge object for the start of the launch window.

-end

Gets the launch_window_edge object for the end of the launch window.

-transparent

Gets the launch_window_edge object for the start of the launch window, for a transparent path from the specified input to the specified output. Must be used together with the **-input** option.

-output_rising *output_port*

Gets the launch_window_edge object for logic 1 output data.

-output_falling *output_port*

Gets the launch_window_edge object for logic 0 output data.

-clock_domain *model_clock_domain*

Specifies the clock domain of the output port.

-input ***input_port***

Gets the launch_window_edge object for the transparent path from the specified input to the specified output (used only with the **-transparent** option).

DESCRIPTION

The **extract_model** command generates a path database from the current design. You can use information from this database to generate a timing model in your own proprietary format. The following commands are available for getting the path database information:

- get_model_clock_domains
- get_capture_window_edges
- get_launch_window_edges
- get_model_delay_arcs
- get_attribute

The **get_launch_window_edges** command gets the starting or ending launch window edge object associated with either a rising or falling transition at a specified output, for a specified launch clock domain.

The launch window is the time span during which the output data is guaranteed to be valid and stable. The start of the launch window is the latest time at which the specified output transition can occur, when the output data is first guaranteed to be valid. The end of the launch window is the earliest time that the output data can change and become invalid.

The command returns a collection of launch_window_edge objects. If no appropriate launch_window_edge object can be found, a zero-length collection is returned.

For a path that starts at a sequential element inside the timing model, you specify the window edge (start or end), output port, output edge type (rising or falling), and launch clock domain.

For a transparent path from input to output, you must use the **-transparent** and **-input** options, and the command returns the launch_window_edge object for the start of the launch window. You specify the input port, output port, output edge type (rising or falling), and launch clock domain.

A launch_window_edge object in the collection has the following attributes:

- output_departure_time
- model_delay_arc
- input_clock_domain (for a transparent path only)

After using the **get_launch_window_edges** command, you can iterate through the collection with the **foreach_in_collection** command and get the launch_window_edge attributes by using the **get_attribute** command.

COMMAND PHASING

The **get_launch_window_edges** command can only be executed after path tracing is complete.

EXAMPLES

The following command gets the launch_window_edge object for the starting edge of the stable-low launch window at output result[7], resulting from a transparent path starting from input data[7]. The command sets the variable "edges2" to the resulting collection.

```
nt_shell> set edges2 [get_launch_window_edges \
          -transparent -output_falling "result[7]" \
          -clock_domain $clock_domain -input "data[7]"]
```

SEE ALSO

collections(2)
extract_model(2)
get_attribute(2)
get_model_clock_domains(2)
get_capture_window_edges(2)
get_model_delay_arcs(2)

get_lib_cells

Creates a collection of library cells from libraries loaded into NanoTime. You can assign these library cells to a variable or pass them into another command.

SYNTAX

```
collection get_lib_cells
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-filter expression]
   patterns | -of_objects objects
```

Data Types

```
   expression       string
   patterns         list
   objects          list
```

ARGUMENTS

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ?

wildcard characters.

-filter *expression*

Filters the collection with *expression*. For any library cells that match *patterns* (or *objects*), the expression is evaluated based on the library cell attributes. If the expression evaluates to **true**, the library cell is included in the result.

-of_objects *objects*

Creates a collection of library cells that are referenced by the specified cells or own the specified library pins. In this case, each object is either a named library pin or netlist cell, or a library pin collection or a netlist cell collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both.

patterns

Matches library cell names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type lib_cell.

DESCRIPTION

The **get_lib_cells** command creates a collection of library cells from libraries currently loaded into NanoTime that match certain criteria. The command returns a collection if any library cells match the *patterns* or *objects* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_lib_cells** command at the command prompt, or you can nest it as an argument for another command (such as the **query_objects** command). In addition, you can assign the **get_lib_cells** command result to a variable.

When issued from the command prompt, the **get_lib_cells** command behaves as though a **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_lib_cells** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command options (for example, if you want to display the object class), use the **get_lib_cells** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_lib_cells** command is not phase-restricted.

EXAMPLES

The following command queries all library cells that are in the misc_cmos library and begin with AN2. Although the output looks like a list, it is just a display.

```
nt_shell> get_lib_cells misc_cmos/AN2*
{"misc_cmos/AN2", "misc_cmos/AN2P"}
```

The following command shows one way to determine the library cell used by a particular cell.

```
nt_shell> get_lib_cells -of_objects [get_cells o_reg1]
{"misc_cmos/FD2"}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_cells(2)
query_objects(2)
collection_result_display_limit(3)
```

get_lib_pins

Creates a collection of library cell pins from libraries loaded into NanoTime. You can assign these library cell pins to a variable or pass them into another command.

SYNTAX

```
collection get_lib_pins
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-filter expression]
   patterns | -of_objects objects
```

Data Types

```
   expression       string
   patterns         list
   objects          list
```

ARGUMENTS

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ?

wildcard characters.

-filter *expression*

Filters the collection with *expression*. For any library cell pins that match *patterns* (or *objects*), the expression is evaluated based on the library cell pin attributes. If the expression evaluates to **true**, the library cell pin is included in the result.

-of_objects *objects*

Creates a collection of library cell pins referenced by the specified netlist pins or owned by the specified library cells. In this case, each object is either a named library cell or netlist pin, or a library cell collection or netlist pin collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both.

patterns

Matches library cell pin names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type lib_pin.

DESCRIPTION

The **get_lib_pins** command creates a collection of library cell pins from libraries currently loaded into NanoTime that match certain criteria. The command returns a collection if any library cell pins match the *patterns* or *objects* and pass the filter (if specified). If no objects matched the criteria, the empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_lib_pins** command at the command prompt, or you can nest it as an argument to another command (such as the **query_objects** command). In addition, you can assign the **get_lib_pins** command result to a variable.

When issued from the command prompt, the **get_lib_pins** command behaves as though a **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed. You can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_lib_pins** command provides a fast, simple way to display cells in a collection. However, if you want the flexibility provided by the **query_objects** command options (for example, if you want to display the object class), use the **get_lib_pins** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_lib_pins** command is not phase-restricted.

EXAMPLES

The following command queries all pins of the AN2 library cell in the misc_cmos library. Although the output looks like a list, it is just a display.

```
nt_shell> [get_lib_pins misc_cmos/AN2/*
{"misc_cmos/AN2/A", "misc_cmos/AN2/B", "misc_cmos/AN2/Z"}
```

The following command shows one way to find out how the library pin is used by a particular pin in the netlist.

```
nt_shell> get_lib_pins -of_objects o_reg1/Q
{"misc_cmos/FD2/Q"}
```

SEE ALSO

```
collections(2)
filter_collection(2)
get_libs(2)
get_lib_cells(2)
query_objects(2)
collection_result_display_limit(3)
```

get_lib_topology

Creates a collection of lib_topology objects.

SYNTAX

```
collection get_lib_topology
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   patterns
```

Data Types

```
   expression       string
   patterns         list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any lib_topology objects that otherwise match, the expression is evaluated based on the attributes of the lib_topology object. The lib_topology object is included in the resulting collection only if the expression is true.

-quiet

Suppresses warning and error messages when no objects match. Syntax error messages are not suppressed.

-regexp

Modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. You can use **-regexp** only when you also use **-filter**.

-nocase

When combined with **-regexp**, makes matches case-insensitive. You can use **-nocase** only when you also use **-regexp** and **-filter**.

patterns

Specifies the lib_topology objects to include in the collection, specified as *library_name* [*hierarchy_separator*] *pattern*. The hierarchy separator is the period character unless changed with the **hierarchy_separator** variable.

DESCRIPTION

The **get_lib_topology** command creates a collection of lib_topology objects. The lib_topology objects can be in the common library, in the global library, or in libraries that have been read with the **read_topology_library** command. You can operate on this collection with the **set_search_enabled** or **remove_search_enabled** command.

You must specify one or more patterns in the **get_lib_topology** command in the following form:

library_name [*hierarchy_separator*] *pattern*

By default, the hierarchy separator is the period character. Thus, to create a collection containing the mux3 topology in the user2 library:

nt_shell> **set mylibs [get_lib_topology user2.mux3]**

You must specify a single topology library explicitly, but you can use a pattern with wildcard characters for the lib_topology name.

COMMAND PHASING

The **get_lib_topology** command can be executed in any state after "netlist-linked" up to and including "paths-traced."

EXAMPLES

The following command enables searching of the mux3 topology in the user2 topology library.

```
nt_shell> set_search_enabled [get_lib_topology user2.mux3]
```

The following command creates a collection of global library topologies for which searching is currently enabled:

```
nt_shell> get_lib_topology \
          -filter "search_enabled==true" global.*
```

The following command creates a collection of all the global lib_topology objects that exist in the design (those that have a matched_count attribute greater than zero):

```
nt_shell> set mycoll2 [get_lib_topology \
          -filter "matched_count>0" global.*]
```

SEE ALSO

```
check_topology(2)
list_lib_topology(2)
list_topology_library(2)
match_topology(2)
read_topology_library(2)
set_search_enabled(2)
hierarchy_separator(3)
```

get_libs

Creates a collection of libraries loaded into NanoTime. You can assign these libraries to a variable or pass them into another command.

SYNTAX

```
collection get_libs
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns | -of_objects objects
```

Data Types

```
   expression       string
   patterns         list
   objects          list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any libraries that match *patterns* (or *objects*), the expression is evaluated based on the library's attributes. If the expression evaluates to **true**, the library is included in the result.

-quiet

Suppresses warning and error messages that occur when no objects match. Syntax error messages are not suppressed.

-regexp

Treats the items in the *patterns* argument as real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters.

patterns

Causes the command to match library names against the list of patterns. Patterns can include the wildcard characters * and ? or regular expressions, based on the **-regexp** option. Patterns can also include collections of type **lib**.

-of_objects ***objects***

Creates a collection of libraries that contain the specified objects. In this case, each object is either a named library cell or a library cell collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both.

DESCRIPTION

The **get_libs** command creates a collection of libraries from those currently loaded into NanoTime that match certain criteria. The command returns a collection if any libraries match the patterns and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the patterns and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression begins matching at the beginning of an object name and ends matching at the end of an object name. You can widen the search by adding .* to the beginning or end of the expression as needed.

You can use the **get_libs** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the result of the **get_libs** command to a variable.

When issued from the command prompt, the **get_libs** command behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_libs** command provides a fast, simple way to display items in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use the **get_libs** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_libs** command is not phase-restricted.

EXAMPLES

The following command queries all loaded libraries. Although the output looks like a list, it is just a display. Note that a complete listing of libraries is available using the **list_libs** command.

```
nt_shell> get_libs *
{"misc_cmos", "misc_cmos_io"}
```

SEE ALSO

collections(2)
filter_collection(2)
list_libs(2)
query_objects(2)
collection_result_display_limit(3)

get_memory

Creates a collection of memories from the current design. You can assign these memories to a variable or pass them into another command.

SYNTAX

```
collection get_memory
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   patterns
```

Data Types

expression	string
patterns	list

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any memory that matches *patterns*, the expression is evaluated based on the memory's attributes. If the expression evaluates to true, the memory is included in the result.

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

patterns

Matches memory names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type "memory".

DESCRIPTION

The NanoTime for memories **get_memory** command creates a collection of memories in the current design that match certain criteria. The command returns a collection if any memory matches the *patterns* list and passes the filter (if specified). If no objects match the criteria, an empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* list and the argument *expression* of the **-filter** option; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_memory** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the result of the **get_memory** command to a variable.

When issued from the command prompt, the **get_memory** command behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_memory** command provides a fast, simple way to display memories in a collection. However, if you want the flexibility provided by the options of the **query_objects** command (for example, if you want to display the object class), use the **get_memory** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_memory** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command queries all memories beginning with "mode." Although the output looks like a list, it is just a display.

```
nt_shell> get_memory "mode*"
{"mode_read", "mode_write"}
```

The following command queries all memories beginning with "mode" and have their mode set to "read."

```
nt_shell> get_memory "mode*" -filter {mode == read}
{"mode_read"}
```

SEE ALSO

```
create_memory(2)
remove_memory(2)
report_memory(2)
collections(2)
query_objects(2)
collection_result_display_limit(3)
```

get_message_ids

Get application message ids

SYNTAX

```
string get_message_ids [-type severity]
[pattern]

string severity
string pattern
```

ARGUMENTS

-type *severity*

Filter ids based on type (Values: Info, Warning, Error, Severe, Fatal)

pattern

Get IDs matching pattern (default: *)

DESCRIPTION

The **get_message_ids** command retrieves the error, warning and informational messages used by the application. The result of this command is a Tcl formatted list of all message ids. Information about the id can be queried with the **get_message_info** command.

EXAMPLES

The following code finds all error messages and makes the application stop script execution when one of

these messages is encountered.

```
foreach id [get_message_ids -type Error] {
  set_message_info -stop_on -id $id
}
```

SEE ALSO

print_message_info(2)
set_message_info(2)
suppress_message(2)

get_message_info

Returns information about diagnostic messages.

SYNTAX

```
integer get_message_info
   [-error_count | -warning_count | -info_count
     | -limit l_id | -occurrences o_id | -suppressed s_id | -id i_id]
```

Data Types

```
   l_id      string
   o_id      string
   s_id      string
   i_id        string
```

ARGUMENTS

-error_count

Returns the number of error messages issued so far.

-warning_count

Returns the number of warning messages issued so far.

-info_count

Returns the number of informational messages issued so far.

-limit *l_id*

Returns the current user-specified limit for a given message ID. The limit was set with the **set_message_info** command.

-occurrences *o_id*

Returns the number of occurrences of a given message ID.

-suppressed *s_id*

Returns the number of times a message was suppressed either using **suppress_message** or due to

exceeding a user-specified limit.

-id *i_id*

Returns information about the specified message. The information is returned as a Tcl list compatible with the array set command.

DESCRIPTION

The **get_message_info** command retrieves information about error, warning, and informational messages. For example, if the following message is generated, information about it is recorded:

Error: unknown command 'wrong_command' (CMD-005)

It is useful to be able to retrieve recorded information about generated diagnostic messages. For example, you can stop a script after a certain number of errors have occurred, or monitor the number of messages issued by a single command.

You can also find out how many times a specific message has occurred, or how many times it has been suppressed. Also, you can find out if a limit has been set for a particular message ID.

EXAMPLES

The following example uses the **get_message_info** command to count the number of errors that occurred during execution of a specific command, and to return from the procedure if the error count exceeds a given amount:

```
prompt> proc \
  do_command {limit} {
    set current_errors [get_message_info -error_count]
    command
    set new_errors [get_message_info -error_count]
    if {[expr $new_errors - $current_errors] > $limit} {
      return -code error "Too many errors"
    }
    ...
  }
```

The following example uses the **get_message_info** command to retrieve information on the CMD-014 message:

```
prompt> get_message_info -id CMD-014
id CMD-014 severity Error limit 0 occurrences 0 suppressed 0 message
{Invalid %s value '%s' in list.}
```

SEE ALSO

print_message_info(2)
set_message_info(2)
suppress_message(2)
get_message_ids(2)

get_model_clock_domains

Returns a collection of clock domains associated with a specified port, obtained from a path database created by the **extract_model** command.

SYNTAX

```
collection get_model_clock_domains
   -input input_port | -output output_port
```

Data Types

```
   input_port        string
   output_port       string
```

ARGUMENTS

-input *input_port*

Specifies an input port and returns a collection of capture clock domains.

-output *output_port*

Specifies an output port and returns a collection of launch clock domains.

DESCRIPTION

The **extract_model** command generates a path database from the current design. You can use information from this database to generate a timing model in your own proprietary format. The following commands are available for getting the path database information:

- get_model_clock_domains
- get_capture_window_edges
- get_launch_window_edges
- get_model_delay_arcs
- get_attribute

The **get_model_clock_domains** command gets the clock domains associated with a specified port. The command returns a collection of model_clock_domain objects, which are capture clock domains if an input port is specified, or launch clock domains if an output port is specified. If no appropriate model_clock_domain objects are found, the command returns a zero-length collection.

A model_clock_domain object has the following attributes:

- clock (string, the clock object name)
- transition_direction (string, rising or falling)

After using the **get_model_clock_domains** command, you can iterate through the collection with the **foreach_in_collection** command and get the model_clock_domain attributes by using the **get_attribute** command.

COMMAND PHASING

The **get_model_clock_domains** command can only be executed after path tracing is complete.

EXAMPLES

The following example gets the clock domains associated with the port "data[12]" when it operates as an output. It sets the variable "domains1" to the resulting collection.

```
nt_shell> set domains1 [get_model_clock_domains -output "data[12]"]
```

SEE ALSO

collections(2)
extract_model(2)
get_attribute(2)
get_capture_window_edges(2)
get_launch_window_edges(2)
get_model_delay_arcs(2)

get_model_delay_arcs

Returns a collection of combinational delay arcs between a specified pair of input and output ports, obtained from a path database created by the **extract_model** command.

SYNTAX

```
collection get_model_delay_arcs
    -min | -max
    -input_rising input_port | -input_falling input_port
    -output_rising output_port | -output_falling output_port
```

Data Types

```
input_port        string
output_port       string
```

ARGUMENTS

-min

Gets minimum-delay timing arcs.

-max

Gets maximum-delay timing arcs.

-input_rising *input_port*

Selects arcs with the signal rising at the specified input port.

-input_falling *input_port*

Selects arcs with the signal falling at the specified input port.

-output_rising *output_port*

Selects arcs with the signal rising at the specified output port.

-output_falling *output_port*

Selects arcs with the signal falling at the specified output port.

DESCRIPTION

The **extract_model** command generates a path database from the current design. You can use information from this database to generate a timing model in your own proprietary format. The following commands are available for getting the path database information:

- get_model_clock_domains
- get_capture_window_edges
- get_launch_window_edges
- get_model_delay_arcs
- get_attribute

The **get_model_delay_arcs** command gets the combinational delay arcs from a specified input to a specified output, having specified edge types (either rising or falling) at the input and output. The command returns a collection of one or more model_delay_arc objects, depending on the paths in the database. If no appropriate model_delay_arc objects are found, the command returns a zero-length collection.

A model_delay_arc object has the following attributes:

- delay
- clock_cycle_count
- load_coefficient
- slope_coefficient
- nominal_load
- nominal_slope
- is_unate
- path

After using the **get_model_delay_arcs** command, you can iterate through the collection with the **foreach_in_collection** command and get the model_delay_arc attributes by using the **get_attribute** command.

COMMAND PHASING

The **get_model_delay_arcs** command can only be executed after path tracing is complete.

EXAMPLES

The following command gets the combinational minimum-delay arcs from input "data[3]" rising to output "result[7]" falling.

```
nt_shell> set arcs [get_model_delay_arcs -min \
          -input_rising "data[3]" \
          -output_falling "result[7]"]
```

SEE ALSO

collections(2)
extract_model(2)
get_attribute(2)
get_model_clock_domains(2)
get_capture_window_edges(2)
get_launch_window_edges(2)

get_nets

Creates a collection of nets from the netlist. You can assign these nets to a variable or pass them into another command.

SYNTAX

```
collection get_nets
   [-hierarchical]
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-top_net_of_hierarchical_group]
   [-segments]
   [-boundary_type lower | upper | both]
   patterns | -of_objects objects
```

Data Types

```
   expression     string
   patterns       list
   objects        list
```

ARGUMENTS

-hierarchical

Searches for nets level by level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a net block1.muxsel, a hierarchical search finds it using "muxsel." You cannot use the **-hierarchical** option with the **-of_objects** option.

-filter *expression*

Filters the collection with *expression*. For any nets that match *patterns* (or *objects*), the expression is evaluated based on the net attributes. If the expression evaluates to **true**, the net is included in the result.

-quiet

Suppresses warning and error messages that occur when no objects match. Syntax error messages are

not suppressed.

-regexp

Treats the items in the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters.

-top_net_of_hierarchical_group

Keeps only the top net of a hierarchical group. When more than one hierarchical net of the same group is specified (local nets at various hierarchical levels of the same physical net), only the net closest to the top of the hierarchy is saved with the collection. In the case of multiple nets at the same level, the first net specified is kept. Although this option can be used when there are multiple nets specified, it is best used in combination with the **-segments** option and with a single net.

-segments

Returns all global segments for given nets. This modifies the initial search that matches *patterns* or *objects*. It is applied after filtering, but before the top net of a hierarchical group is determined. For each net, all global segments that are part of that net are added to the result collection. Global net segments are all those physically connected across all hierarchical boundaries. Although this option can be used with other options and when there are multiple nets specified, it is best used with a single net. When combined with the **-top_net_of_hierarchical_group** option, you can isolate the highest net segment of a physical net.

-boundary_type lower | upper | both

Specifies what to do when getting nets of boundary pins. This option requires the **-of_objects** option. The allowed values are **lower**, **upper**, or **both**, meaning the net inside the hierarchical block, outside the hierarchical block, or both nets, respectively. The option only affects hierarchical pins. Note that the **upper** setting is less useful, as getting pins of a hierarchical net returns the pin outside the hierarchical block, and a subsequent **get_nets** command only finds the upper hierarchical net. Using the **upper** setting on a hierarchical pin is the same as omitting the option.

patterns

Matches net names against patterns. Patterns can include the wildcard characters * and ? or regular expressions, based on the **-regexp** option. Patterns can also include collections of type **net**.

-of_objects *objects*

Creates a collection of nets connected to the specified objects. Each object is either a named pin or a pin collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both. In addition, you cannot use the **-hierarchical** option with the **-of_objects** option.

DESCRIPTION

The **get_nets** command creates a collection of nets in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any nets match the patterns or objects and pass the filter (if specified). If no objects match the criteria, the empty collection is returned.

If any patterns (or objects) fail to match any objects and the current design is not linked, the design is automatically linked.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_nets** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the result of the **get_nets** command to a variable.

When issued from the command prompt, the **get_nets** command behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_nets** command provides a fast, simple way to display nets in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use the **get_nets** command as an argument to the **query_objects** command.

For more information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_nets** command is not phase-restricted.

EXAMPLES

The following command queries the nets that begin with "d" in block xl3. Although the output looks like a list, it is just a display.

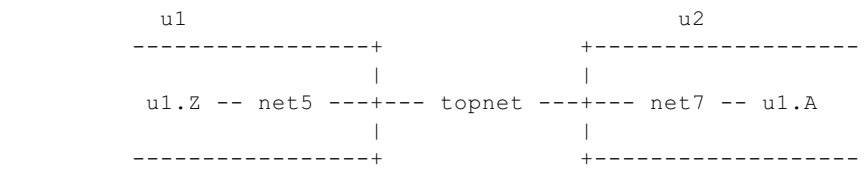
```
nt_shell> get_nets xl3.d*
{"xl3.d", "xl3.dn"}
```

The following example shows that with a collection of pins, you can query the nets connected to those pins.

```
nt_shell> current_instance xl3
xl3
```

```
nt_shell> set pinsel [get_pins {xi2.a xi3.a}]
{"xl3.xi2.a", "xl3.xi3.a"}
nt_shell> query_objects [get_nets -of_objects $pinsel]
{"xl3.qn", "xl3.q"}
```

This example shows how to use the **-top_net_of_hierarchical_group** and **-boundary_type** options. Given the following circuit:



There are hierarchical cells u1 and u2 at this level, connected by a net *topnet*. Within u1 is a pin u1.Z driving *net5*, and within u2 is a pin u1.A being driven by *net7*. These three nets are physically the same net. Assume these are the only nets in the design. Notice the difference in the results of the following two **get_nets** commands:

```
nt_shell> get_nets * -hierarchical
{"topnet", "u1.net5", "u2.net7"}
nt_shell> get_nets * -hierarchical -top_net_of_hierarchical_group
{"topnet"}
```

Assume that at the top level, topnet is connected to pins u2.in and u1.out. Here are some examples of the **-boundary_type** option. Notice that in this case, the **upper** setting does not add value.

```
nt_shell> get_nets -of_objects u2.in
{"topnet"}
nt_shell> get_nets -of_objects u2.in -boundary_type upper
{"topnet"}
nt_shell> get_nets -of_objects u2.in -boundary_type lower
{"u2.net7"}
nt_shell> get_nets -of_objects u2.in -boundary_type both
{"u2.net7", "topnet"}
nt_shell> get_nets -boundary_type lower -of_objects \
             [get_pins -of_objects [get_nets topnet]]
{"u1.net5", "u2.net7"}
```

SEE ALSO

collections(2)
filter_collection(2)
get_pins(2)
link_design(2)
query_objects(2)
collection_result_display_limit(3)

get_pins

Creates a collection of pins from the netlist. You can assign these pins to a variable or pass them into another command.

SYNTAX

```
collection get_pins
   [-hierarchical]
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   [-leaf]
   patterns | -of_objects objects
```

Data Types

```
   expression     string
   patterns       list
   objects        list
```

ARGUMENTS

-hierarchical

Searches for pins level by level relative to the current instance. The full name of the object at a particular level must match the patterns. The search is similar to the UNIX **find** command. For example, if there is a pin block1.adder.D[0], a hierarchical search finds it using the "adder.D[0]" string. You cannot use the **-hierarchical** option with the **-of_objects** option.

-filter *expression*

Filters the collection with *expression*. For any pins that match *patterns* (or *objects*), the expression is evaluated based on the pin attributes. If the expression evaluates to **true**, the pin is included in the result.

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters.

-leaf

You can use this option only with the **-of_objects** option. For any nets in the *objects* argument of the **-of_objects** option, only pins on leaf cells connected to those nets are included in the collection. In addition, hierarchical boundaries are crossed to find pins on leaf cells.

patterns

Matches pin names against patterns. Patterns can include the wildcard xxx characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type "pin."

-of_objects ***objects***

Creates a collection of pins connected to the specified objects. Each object is a named cell, net, cell collection, or net collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both. In addition, you cannot use the **-hierarchical** option with the **-of_objects** option.

DESCRIPTION

The **get_pins** command creates a collection of pins in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any pins match the *patterns* or *objects* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

When used with the **-of_objects** option, the **get_pins** command searches for pins connected to any cells or nets specified in *objects*. For net objects, there are two variations of pins that can be considered. By default, only pins connected to the net at the same hierarchical level are considered. When combined with the **-leaf** option, only pins connected to the net that are on leaf cells are considered. In this case, hierarchical boundaries are crossed to find pins on leaf cells. Note that the **-leaf** option has no effect on the pins of cells.

If no *patterns* (or *objects*) match any objects, and the current design is not linked, the design is automatically linked.

When a cell has bus pins, **get_pins** can find them in several ways. For example, if cell u1 has a bus "A" with indexes 2 to 0, and the bus naming style for your design is "%s[%d]", then to find these pins, you

can use "u1.A[*]" as the pattern. You can also find the same three pins with "u1.A" as the pattern.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_pins** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the result of the **get_pins** command to a variable.

When issued from the command prompt, the **get_pins** command behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_pins** command provides a fast, simple way to display nets in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use the **get_pins** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page.

COMMAND PHASING

The **get_pins** command is not phase-restricted.

EXAMPLES

The following command queries all the pins of a transistor cell. Although the output looks like a list, it is just a display.

```
nt_shell> get_pins x1.mn1.*
{"x1.mn1.d", "x1.mn1.g", "x1.mn1.s", "x1.mn1.b"}
```

The following example queries the pins connected to the cells in a collection.

```
nt_shell> set csel [get_cells xl3]
{"xl3"}
nt_shell> query_objects [get_pins -of_objects $csel]
{"xl3.d", "xl3.g", "xl3.q"}
```

The following example shows the difference between getting local pins of a net and leaf pins of a net. In this example, NET1 is connected to i2.a and reg1.QN. Cell i2 is hierarchical. Within i2, port a is connected to the pins U1.A and U2.A.

```
nt_shell> get_pins -of_objects [get_nets NET1]
{"i2.a", "reg1.QN"}
nt_shell> get_pins -leaf -of_objects [get_nets NET1]
{"i2.U1.A", "i2.U2.A", "reg1.QN"}
```

```
nt_shell> get_pins -of_objects [get_nets out1]
{"xl3.q", "c12.term1"}
nt_shell> get_pins -leaf -of_objects [get_nets out1]
{"c12.term1", "xl3.xi3.mp1.g", "xl3.xi3.mn1.g", "xl3.xi2.mp1.s",
"xl3.xi2.mn1.d"}
```

SEE ALSO

collections(2)
filter_collection(2)
get_cells(2)
link_design(2)
query_objects(2)
collection_result_display_limit(3)

get_ports

Creates a collection of ports from the current design. You can assign these ports to a variable or pass them into another command.

SYNTAX

```
collection get_ports
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
   [-exact]
   patterns | -of_objects objects
```

Data Types

```
   expression     string
   patterns       list
   objects        list
```

ARGUMENTS

-filter *expression*

Filters the collection with *expression*. For any ports that match *patterns*, the expression is evaluated based on the port attributes. If the expression evaluates to **true**, the port is included in the result.

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

-regexp

Views the *patterns* argument as a set of real regular expressions rather than simple wildcard patterns. Also, modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. The **-regexp** and **-exact** options are mutually exclusive.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase**

option only when you also use the **-regexp** option.

-exact

Disables simple pattern matching. This is used when searching for objects that contain the * and ? wildcard characters. The **-exact** and **-regexp** options are mutually exclusive.

patterns

Matches port names against patterns. Patterns can include the wildcard characters "*" and "?" or regular expressions, based on the **-regexp** option. Patterns can also include collections of type "port."

-of_objects ***objects***

Creates a collection of ports connected to the specified objects. Each object is a named net collection. The **-of_objects** and *patterns* options are mutually exclusive; you must specify one, but not both.

DESCRIPTION

The **get_ports** command creates a collection of ports in the current design, relative to the current instance, that match certain criteria. The command returns a collection if any ports match the *patterns* or *objects* and pass the filter (if specified). If no objects match the criteria, the empty string is returned.

Regular expression matching is the same as in the Tcl **regexp** command. When using the **-regexp** option, take care in the way you quote the *patterns* and filter *expression*; use rigid quoting with curly braces around regular expressions. Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expressions as needed.

You can use the **get_ports** command at the command prompt, or you can nest it as an argument to another command (for example, the **query_objects** command). In addition, you can assign the result of the **get_ports** command to a variable.

When issued from the command prompt, the **get_ports** command behaves as though the **query_objects** command has been called to display the objects in the collection. By default, a maximum of 100 objects is displayed; you can change this maximum using the **collection_result_display_limit** variable.

The "implicit query" property of the **get_ports** command provides a fast, simple way to display nets in a collection. However, if you want the flexibility provided by the **query_objects** options (for example, if you want to display the object class), use the **get_ports** command as an argument to the **query_objects** command.

For information about collections and the querying of objects, see the **collections** man page. In addition, see the man pages for the **all_inputs** and **all_outputs** commands, which also create collections of ports.

COMMAND PHASING

The **get_ports** command is not phase-restricted.

EXAMPLES

The following command queries all input ports beginning with "mode." Although the output looks like a list, it is just a display.

```
nt_shell> get_ports "mode*" -filter {direction == in}
{"mode[0]", "mode[1]", "mode[2]"}
```

The following command sets the drive resistance of all ports beginning with "in" to 100.0.

```
nt_shell> set_drive 100.0 [get_ports in*]
```

The following command reports ports connected to nets matching the "bidir*" pattern.

```
nt_shell> report_port [get_ports -of_objects [get_nets "bidir*"]]
```

SEE ALSO

all_inputs(2)
all_outputs(2)
set_port_direction(2)
collections(2)
filter_collection(2)
query_objects(2)
collection_result_display_limit(3)

get_simulations

Creates a collection of simulation objects in the design based on a specified list of input or output pins or nets.

SYNTAX

```
status get_simulations
   -input pin_or_net
   -output pin_or_net
```

Data Types

```
   pin_or_net      list
```

ARGUMENTS

-input *pin_or_net*

Gets only the simulation objects in the design that have the specified pins or nets at the inputs of the simulation region.

-output *pin_or_net*

Gets only the simulation objects in the design that have the specified pins or nets at the outputs of the simulation region.

DESCRIPTION

The **get_simulations** command creates a collection of simulation objects in the design. The collection includes only the simulation objects that have the specified pins or nets at the inputs or outputs of the simulation region. A simulation object is a region in the design selected for simulation using dynamic delay simulation. The region is specified with the **mark_simulation** command and recognized by the **check_design** command.

You can set a variable to the collection and then perform operations on the collection using commands

such as **report_simulation** and **set_simulation_attributes**.

To create a collection of all the simulation objects in the design, use the **all_simulations** command.

COMMAND PHASING

The **get_simulations** command must be executed after the **check_design** command but before the **trace_paths** command.

EXAMPLES

The following command generates a report on the simulation object in the design that has net n18 at the output of the simulation region:

```
nt_shell> report_simulation \
          [get_simulations -output [get_nets n18]]

Summary of Simulations

                            Num     Num     Num   Num
Outputs                     Outputs Inputs  Nets  TX
--------------------------- ------- ------- ----- -----
 ...
```

The following command sets the variable mysim to the simulation object that has net n18 at the output of the simulation region:

```
nt_shell> set mysim \
          [get_simulations -output [get_nets n18]]
```

SEE ALSO

```
all_simulations(2)
mark_simulation(2)
report_simulation(2)
set_simulation_attributes(2)
```

get_timing_checks

Creates a collection of timing checks.

SYNTAX

```
collection get_timing_checks
   [-from from_object
      | -rise_from from_object
      | -fall_from from_object]
   [-to to_object
      | -rise_to to_object
      | -fall_to to_object]
   [-target target_object
      | -rise_target target_object
      | -fall_target target_object]
   [-trigger trigger_object
      | -rise_trigger trigger_object
      | -fall_trigger trigger_object]
   [-setup]
   [-hold]
   [-to_not_target]
   [-filter expression]
   [-quiet]
```

Data Types

```
   from_object          list
   to_object            list
   target_object        list
   trigger_object       list
   expression           string
```

ARGUMENTS

-from *from_object*

Specifies the clock or "reference" port or pin of the timing check. This is the "related" pin of the timing check.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising transitions at the clock pin.

-fall_from ***from_object***

Similar to the **-from** option, but applies only to falling transitions at the clock pin.

-to ***to_object***

Specifies the data pin or port in the current design where the timing check applies. This is the "constrained" or "checked" pin of the timing check. Specifying this option without the **-to_not_target** option restricts the search to intrace timing checks which are evaluated during path tracing.

-rise_to ***to_object***

Similar to the **-to** option, but applies only to rising signals at the data pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling signals at the data pin.

-target ***target_object***

The target port or pin at the output of the simulation unit related to the timing check, where path tracing ends, which might not be the same as the timing check point ("to" object).

-rise_target ***target_object***

Similar to the **-target** option, but applies only to rising signals at the target object.

-fall_target ***target_object***

Similar to the **-target** option, but applies only to falling signals at the target object.

-trigger ***trigger_object***

The trigger port or pin at the input of the simulation unit related to the timing check, where path tracing starts.

-rise_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to rising signals at the trigger object.

-fall_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to falling signals at the trigger object.

-setup

Gets only the setup check.

-hold

Gets only the hold check.

-to_not_target ***to_object***

Specifies that the "to" object is different from the "target" object. In the absence of this option, a "to" object specified by itself is considered to be both the timing check point and the "target" object at the output of the simulation unit, where path tracing ends. This option must be specified to retrieve data-to-data (including clock-to-clock) timing checks, which are evaluated after path tracing.

-filter ***expression***

Filters the collection with the given expression. For any timing check that meets the selection criteria, the expression is evaluated based on the timing check attributes. If the expression evaluates to **true**, the timing check is included in the result.

-quiet

Suppresses warning messages.

DESCRIPTION

Use the **get_timing_checks** command to create a collection of existing timing checks, whether NanoTime created them automatically or you created them manually. You can then modify the timing check properties with the **set_timing_check_attributes** command.

The **get_timing_checks** command can specify up to four timing check characteristics:

```
o "from" object: clock/reference pin of the timing check
o "to" object: data pin of the timing check
o "trigger" object: input pin of the simulation unit
o "target" object: output pin of the simulation unit
```

NanoTime performs timing checks by considering the arrival times at the "from" and "to" points. The "from" object is a port or pin where there is a clock signal or the "related" pin of a data-to-data timing check. This is sometimes called the "reference" pin of the timing check. The "to" object is a port or pin where there the data arrival timing check is performed. This is sometimes called the "constrained pin" of the timing check.

The "from" and "to" points specify where the timing checks are performed, whereas the "trigger" and "target" points specify where path tracing is performed. To narrowly define the timing checks retrieved by the **get_timing_checks** command, you can specify all four objects, the "from," "to," "trigger," and "target" objects. Specifying fewer objects results in a larger collection of timing checks. For example, if you specify only the "from" object, the collection contains all timing checks where the "from" object is the clock. Those timing checks might have different "to," "trigger," and "target" objects.

By default, NanoTime assumes that a "to" object specified by itself is also the "target" object, so it looks for timing checks where the "to" object is the constrained pin of the timing check and is also the endpoint of path tracing. This type of timing check is associated with a pin, not a timing arc.

If the timing check is associated with a timing arc rather than a pin, the "target" object is not the same as the "to" object. To look for this type of timing check, either specify the "target" object explicitly or use the **-to_not_target** option. NanoTime looks for timing arcs where the timing check is performed at the "to" object, but path tracing stops at the "target" object (different from the "to" object). To retrieve data checks which are evaluated after path tracing, use the **-to_not_target** option along with the **-to** or **-from** options, or both.

In a master-slave flip-flop structure, the trigger object is the latch node of the master latch and the target is the latch node of the slave latch. The simulation unit consists of the circuitry from the trigger to the target. However, unlike a simple latch, the flip-flop captures data on the opening edge rather than the closing edge, and the target pin is nontransparent rather than transparent. The "from" object is the clock pin of the slave latch, and the "to" object (where the data arrival time is measured) is the data input pin of

the slave latch.

Each object specification option can apply to both rising or falling transitions, or only to rising or only to falling transitions at that object, depending on the option used.

If you are getting a timing check created earlier by the **create_timing_check** command, the "from" and "to" objects in the **get_timing_checks** command must match the ones used in the original **create_timing_check** command, as well as the **-setup** and **-hold** options, if used originally.

To extract information about the timing checks in the collection generated by **get_timing_checks**, use the **get_attribute** command, in the following form:

nt_shell> get_attribute *collection_name attr_name*

To list the attributes associated with a timing check, use the **list_attributes** command.

COMMAND PHASING

The **get_timing_checks** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example gets a timing check previously set with the **create_timing_check** command, sets a variable to a collection containing that check, and extracts the check value from the collection. The "from" and "to" specifications must exactly match the ones used in the original **create_timing_check** commands.

```
nt_shell> set mycheck [get_timing_checks \
          -rise_from [get_ports CLK] \
          -to [get_ports DATA1]]

nt_shell> get_attribute $mycheck check_value
```

SEE ALSO

create_timing_check(2)
set_data_check(2)
filter_collection(2)
get_attribute(2)
list_attributes(2)
remove_timing_check(2)
set_timing_check_attributes(2)

get_timing_paths

Creates a collection of timing paths for custom reporting and other processing. You can assign these timing paths to a variable or pass them to another command.

SYNTAX

```
collection get_timing_paths
   -min | -max | -path_id path_id_list | -model_paths
   [-clock_only]
   [-by  slack | delay]
   [-from from_list
      | -rise_from from_list
      | -fall_from from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to to_list
      | -fall_to to_list]
   [-slack_less_than slack_limit]
   [-slack_greater_than slack_limit]
   [-nworst paths_per_endpoint]
   [-npaths_per_startpoint paths_per_startpoint]
   [-max_paths max_path_count]
   [-quiet]
```

Data Types

path_id_list	list
from_list	list
through_list	list
rise_through_list	list
fall_through_list	list
to_list	list
slack_limit	int
paths_per_endpoint	int
paths_per_startpoint	int
max_path_count	int

ARGUMENTS

-min

Gets the shortest (minimum-delay) paths. Exactly one of the **-min**, **-max**, **-path_id**, or **-model_paths** options must be used.

-max

Gets the longest (maximum-delay) paths.

-path_id ***path_id_list***

Specifies the ID numbers of the paths to get. This option causes the command to get only those paths that have the specified ID numbers. Path ID numbers are displayed by the **report_paths** command when the **-show_path_id** option is used.

-model_paths

Gets the paths that have been used by the **extract_model** command to generate a timing model. If the **-clock_only** option is used, the command gets only the clock paths used for modeling. Otherwise, it gets only the data paths used for modeling.

-clock_only

Gets paths created during clock path tracing only. The default is to get paths created during data path tracing only.

-by slack | delay

Specifies whether to get the paths with the worst **slack** or worst **delay**. For **slack** (the default), the command gets the paths with the least slack. For **delay**, it gets the paths with the worst delay. These paths can be different because of differences in clock arrival times.

-from ***from_list***

Specifies a list of pins, ports, nets, or clocks to be considered path startpoints. Only paths that start at the specified objects are reported. These objects are typically the input ports. If you specify a clock, all startpoints clocked by the specified clock are considered.

-rise_from ***from_list***

Same as the **-from** option, except that the transition at the startpoint must be a rising edge. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from ***from_list***

Same as the **-from** option, except that the transition at the startpoint must be a falling edge. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass. Only paths that go through the specified objects are considered.

-rise_through ***rise_through_list***

Same as the **-through** option, except that the signal transition arriving at the object must be rising.

-fall_through ***fall_through_list***

Same as the **-through** option, except that the signal transition arriving at the object must be falling.

-to ***to_list***

Specifies a list of pins, ports, nets, or clocks to be considered path endpoints. Only paths that end at the specified objects are reported. These objects are typically output ports and data pins of registers. If you specify a clock, all endpoints constrained by the specified clock are considered.

-rise_to ***to_list***

Same as the **-to** option, except that the transition at the endpoint must be a rising edge. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to ***to_list***

Same as the **-to** option, except that the transition at the endpoint must be a falling edge. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by falling edge of the clock at clock source, taking into account any logical inversions along the clock path.

-slack_less_than ***slack_limit***

Restricts reporting to paths that have a slack less (or more negative) than the specified value. Specify 0.0 to report only the paths that have timing violations.

-slack_greater_than ***slack_limit***

Restricts reporting to paths that have a slack greater (or less negative) than the specified value.

-nworst ***paths_per_endpoint***

Filters the set of paths so that the selected set contains at most *paths_per_endpoint* paths that end at any given endpoint. By default, no such filtering is performed.

-npaths_per_startpoint ***paths_per_startpoint***

Filters the set of paths so that the selected set contains at most *paths_per_startpoint* paths that start at any given startpoint. By default, no such filtering is performed.

-max_paths ***max_path_count***

Specifies the maximum number of paths to get. By default, there is no limit.

-quiet

Suppresses warning and error messages that would otherwise occur when no objects match. Syntax error messages are not suppressed.

DESCRIPTION

The **get_timing_paths** command creates a collection of paths for custom reporting or other operations. The paths are selected from the paths that have previously been created by the **trace_paths** command and stored in the path database.

Use the **foreach_in_collection** command to iterate through the paths in the collection. You can use the **get_attribute** and **collection** commands to obtain information about the paths. Use the **list_attributes** command to see the set of attributes associated with a timing path.

One attribute of a timing path is the *points* collection. A point corresponds to a pin or port along the path. You can iterate through these points with the **foreach_in_collection** command and get attributes on them by using the **get_attribute** command.

For information about creating collections and iterating over the elements of a collection, see the man pages for the **collections** and **foreach_in_collection** commands.

You can restrict the paths that are included in the collection according to startpoint, intermediate point, or endpoint by using the **-from**, **-through**, or **-to** options. You can further restrict the paths according to transition type by using the **-rise_from**, **-fall_to**, and similar options.

If you use multiple **-through** options, the path must pass through each of the specified objects in the order listed. If you specify a list of multiple objects in one **-through** option, the path can pass through any one of those objects. For example, specifying **-from A1 -through {B1 B2} -through {C1 C2} -to D1** means any path that starts at A1, passes through either B1 or B2, then through either C1 or C2, and ends at D1.

COMMAND PHASING

The **get_timing_paths** command can only be executed after path tracing is complete.

EXAMPLES

The following command generates a path-based slack adjustment report for the worst-case (least-slack) hold check stored in the path database.

```
nt_shell> report_pbsa_calculation \
          [get_timing_paths -min -max_paths 1]
```

The following command generates a path-based slack adjustment report for the worst-case (least-slack) setup check stored in the path database that starts at port IN1 and ends at port OUT2.

```
nt_shell> report_pbsa_calculation \
          [get_timing_paths \
          -from [get_ports IN1] \
          -to [get_ports OUT2] \
          -max -max_paths 1]
```

The following procedure prints out the startpoint name, endpoint name, and the slack of the worst long path to each unique endpoint.

```
proc custom_report_worst_path_per_endpoint {} {
```

```
    echo [format "%-20s %-20s %7s" "From" "To" "Slack"]
    echo "-------------------------------------------------"
    foreach_in_collection path [get_timing_paths -max -nworst 1] {
        set slack [get_attribute $path slack]
        set startpoint [get_attribute $path startpoint]
        set sp_obj [get_attribute $startpoint object]
        set endpoint [get_attribute $path endpoint]
        set ep_obj [get_attribute $endpoint object]
        echo [format "%-20s %-20s %s" \
                [get_attribute $sp_obj full_name] \
                [get_attribute $ep_obj full_name]  $slack]
    }
}

nt_shell> custom_report_worst_path_per_endpoint
From                 To                       Slack
-------------------------------------------------
clk1                 out2                 -3.000000
clk1                 out1                 -3.000000
clk1                 xl4.xi2.mn1.g        -1.000000
clk1                 xl4.xi2.mp1.g        -1.000000
clk1                 xl3.xi2.mp1.g        -1.000000
clk1                 xl3.xi2.mn1.g        -1.000000
in0                  xl1.xi2.mn1.g        6.000000
in0                  xl2.xi2.mp1.g        6.000000
in0                  xl2.xi2.mn1.g        6.000000
in0                  xl1.xi2.mp1.g        6.000000
```

You can find some detailed examples of custom timing reports in:

```
$SYNOPSYS_ROOT/auxx/nt/examples/tcl
```

where $SYNOPSYS_ROOT is the installation directory for NanoTime.

SEE ALSO

```
list_attributes(2)
collections(2)
extract_model(2)
foreach_in_collection(2)
get_attribute(2)
report_paths(2)
trace_paths(2)
```

get_topology

Creates a collection of topology objects.

SYNTAX

```
collection get_topology
   -all | -of_objects objects
   [-structure_type type]
   [-filter expression]
   [-quiet]
   [-regexp]
   [-nocase]
```

Data Types

```
   objects         list
   type            string
   expression      string
```

ARGUMENTS

-all

Gets all topology objects in the design, subject to the criteria in the **-structure_type** and **-filter** options. One of the **-all** or **-of_objects** options is required.

-of_objects *objects*

Gets the topologies associated with objects in the given object collection. Objects in the *objects* collection can be nets or leaf-level cells.

-structure_type *type*

Limits the selected topologies to only those of the specified structure type. The allowed values for structure type are **clock_gate**, **cross_coupled**, **cross_coupled_pmos**, **differential_synchronizer**, **feedback**, **flip_flop**, **inverter**, **latch**, **memory_precharge**, **memory_write_circuit**, **mux**, **nand**, **nor**, **power_switch**, **precharge**, **pulldown**, **pullup**, **ram**, **register**, **sense_amp**, **tgate**, **turnoff**, **weak_pullup**, and **xor**.

-filter *expression*

Filters the collection with a logical expression. For any topologies that otherwise match, the expression is evaluated based on the topology's attributes. The topology is included in the result only if the expression is true.

-quiet

Suppresses warning and error messages when no matching objects are found. Syntax error messages are not suppressed.

-regexp

Modifies the behavior of the =~ and !~ filter operators to compare with real regular expressions rather than simple wildcard patterns. You can use the **-regexp** option only with the **-filter** option.

-nocase

When combined with the **-regexp** option, makes matches case-insensitive. You can use the **-nocase** option only with the **-regexp** and **-filter** options.

DESCRIPTION

The **get_topology** command creates a collection of topologies in the current design. Topologies are created by automatic recognition with the **match_topology** command or by manual marking with the **mark_*** commands. The collection created by the **get_topology** command can be reported with the **report_topology** command, erased with the **erase_topology** command, or traversed with the **foreach_in_collection** command. During traversal, you can report each topology object's attributes with the **get_attribute** command.

The **get_topology** command must use one of the **-all** or **-of_objects** options. Use the **-all** option to search through the entire current design. Use the **-of_objects** option to restrict the search to a specified collection of nets or leaf-level cells.

The **-filter** option restricts the collection to structures for which a logical expression is true. For example, to restrict the collection to structures that have been assigned a name with the **-name** option of the **match_topology** or **mark_* command**, use the following expression:

-filter "structure_name == string"

COMMAND PHASING

The **get_topology** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command removes recognition of all latch structures in the design.

```
nt_shell> erase_topology \
```

```
            [get_topology -all -structure_type latch]
```

The following command reports the number of inverters currently recognized or marked in the design.

```
nt_shell> echo [sizeof_collection \
          [get_topology -all -structure_type inverter]]
25
```

The following command generates a list of all latches with the structure name "mylatch1."

```
nt_shell> get_topology -all -structure_type latch \
          -filter "structure_name == mylatch1"
```

SEE ALSO

check_topology(2)
collections(2)
erase_topology(2)
foreach_in_collection(2)
get_attribute(2)
mark_latch(2)
match_topology(2)
report_topology(2)

getenv

Returns the value of a system environment variable.

SYNTAX

```
string getenv
   variable_name
```

Data Types

```
   variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of the environment variable to be retrieved.

DESCRIPTION

The **getenv** command searches the system environment for the specified *variable_name* and sets the result of the command to the value of the environment variable. If the variable is not defined in the environment, the command returns a Tcl error. The command is catchable.

Environment variables are stored in the **env** Tcl array variable. The **getenv**, **setenv**, and **printenv** environment commands are convenience functions to interact with this array.

The application you are running inherited the initial values for environment variables from its parent process (that is, the shell from which you invoked the application). If you set the variable to a new value using the **setenv** command, you see the new value within the application and within any new child processes you initiate from the application using the **exec** command. However, these new values are not exported to the parent process. Further, if you set an environment variable using the appropriate system command in a shell you invoke using the **exec** command, that value is not reflected in the current application.

See the **set**, **unset**, and **printvar** commands for information about working with non-environment

variables.

EXAMPLES

In the following example, **getenv** returns you to your home directory:

```
prompt> set home [getenv "HOME"]
/users/disk1/bill

prompt> cd $home

prompt> pwd
/users/disk1/bill
```

In the following example, **setenv** changes the value of an environment variable:

```
prompt> getenv PRINTER
laser1

prompt> setenv PRINTER "laser3"
laser3

prompt> getenv PRINTER
laser3
```

In the following example, the requested environment variable is not defined. The error message shows that the Tcl variable **env** was indexed with the value UNDEFINED, which resulted in an error. In the second command, **catch** is used to suppress the message.

```
prompt> getenv "UNDEFINED"
Error: can't read "env(UNDEFINED)": no such element in array
        Use error_info for more info. (CMD-013)

prompt> if {[catch {getenv "UNDEFINED"} msg]} {
          setenv UNDEFINED 1
        }
```

SEE ALSO

catch(2)
exec(2)
printenv(2)
printvar(2)
set(2)
setenv(2)
unsetenv(2)
unset(2)

help

Displays quick help for one or more commands.

SYNTAX

```
string help
   [-verbose]
   [-groups]
   [pattern]
```

Data Types

```
   pattern      string
```

ARGUMENTS

-verbose

Displays the command options; for example *command_name* **-help**.

-groups

Displays a list of command groups only.

pattern

Displays commands matching the specified pattern.

DESCRIPTION

The **help** command is used to get quick help for one or more commands or procedures. This is not the same as the **man** command that displays reference manual pages for a command. There are many levels of help.

By typing the **help** command, a brief informational message is printed followed by the available command groups.

List all of the commands in a group by typing the group name as the argument to **help**. Each command is followed by a one-line description of the command.

To get a one-line help description for a single command, type **help** followed by the command name. You can specify a wildcard pattern for the name; for example, all commands containing the string "alias". Use the **-verbose** option to get syntax help for one or more commands. Use the **-groups** option to show only the command groups in the application. This option cannot be combined with any other option.

EXAMPLES

The following example lists the commands by command group:

```
prompt> help
Specify a command name or wild card pattern to get help on individual
commands. Use the '-verbose' option to get detailed option help for a
command. Commands also provide help when the '-help' option is passed to
the command.'

You can also specify a command group to get the commands available in
that group. Available groups are:

  Procedures:               Miscellaneous procedures
  Help:                     Help commands
  Builtins:                 Generic Tcl commands
 ...
```

The following example displays the list of procedures in the Procedures group:

```
prompt> help procedures
 ls                 # List files
 sh                 # Execute a shell command
```

The following example uses a wildcard character to display a one-line description of all commands beginning with *a*:

```
prompt> help a*
 alias              # Create a command which expands to words.
 append             # Builtin
 array              # Builtin
```

This example displays option information for the **source** command:

```
prompt> help -verbose source
 source             # Read a file and execute it as a script
   [-echo]              (Echo all commands)
   [-verbose]           (Display intermediate results)
   file_name            (Script file to read)
```

SEE ALSO

```
man(2)
sh_help_shows_group_overview(3)
```

history

Displays or modifies the commands recorded in the history list.

SYNTAX

```
string history
   [-h]
   [-r]
   [argument_list]
```

Data Types

```
   argument_list      list
```

ARGUMENTS

-h

Displays the history list without the leading numbers. You can use this for creating scripts from existing history. You can then source the script with the **source** command. You can combine this option with only a single numeric argument. Note that this option is a nonstandard extension to Tcl.

-r

Reverses the order of output so that most recent history entries display first rather than the oldest entries first. You can combine this option with only a single numeric argument. Note that this option is a nonstandard extension to Tcl.

argument_list

Additional arguments to **history** (see DESCRIPTION).

DESCRIPTION

The **history** command performs one of several operations related to recently-executed commands

recorded in a history list. Each of these recorded commands is referred to as an "event." The most commonly used forms of the command are described below. You can combine each with either the **-h** or **-r** option, but not both.

- With no arguments, the **history** command returns a formatted string (intended for you to read) giving the event number and contents for each of the events in the history list.
- If a single, integer argument *count* is specified, only the most recent *count* events are returned. Note that this option is a nonstandard extension to Tcl.
- Initially, 20 events are retained in the history list. You can change the length of the history list using the following:

  history keep *count*

Tcl supports many additional forms of the **history** command. See the "Advanced Tcl History" section below.

EXAMPLES

The following examples show the basic forms of the **history** command. The first is an example of how to limit the number of events shown using a single numeric argument:

```
prompt> history 3
     7  set base_name "my_file"
     8  set fname [format "%s.db" $base_name]
     9  history 3
```

Using the **-r** option creates the history listing in reverse order:

```
prompt> history -r 3
     9  history -r 3
     8  set fname [format "%s.db" $base_name]
     7  set base_name "my_file"
```

Using the **-h** option removes the leading numbers from each history line:

```
prompt> history -h 3
set base_name "my_file"
set fname [format "%s.db" $base_name]
history -h 3
```

Advanced Tcl History

The **history** command performs one of several operations related to recently-executed commands recorded in a history list. Each of these recorded commands is referred to as an "event." When specifying an event to the **history** command, the following forms may be used:

- A number, which if positive, refers to the event with that number (all events are numbered starting at 1). If the number is negative, it selects an event relative to the current event; for example, **-1** refers to the previous event, **-2** to the one before that, and so on. Event **0** refers to the current event.

- A string selects the most recent event that matches the string. An event is considered to match the string either if the string is the same as the first characters of the event, or if the string matches the event in the sense of the **string match** command.

The **history** command can take any of the following forms:

history

Same as **history info**, described below.

history add *command [exec]*

Adds the *command* argument to the history list as a new event. If **exec** is specified (or abbreviated), the command is also executed and its result is returned. If **exec** is not specified, an empty string is returned.

history change *newValue [event_number]*

Replaces the value recorded for an event with *newValue*. The *event_number* specifies the event to replace, and defaults to the *current* event (not event **-1**). This command is intended for use in commands that implement new forms of history substitution and want to replace the current event (that invokes the substitution) with the command created through substitution. The return value is an empty string.

history clear

Erases the history list. The current keep limit is retained. The history event numbers are reset.

history event *[event_number]*

Returns the value of the event given by *event_number*. The default value of *event_number* is **-1**.

history info *[count]*

Returns a formatted string, giving the event number and contents for each of the events in the history list except the current event. If *count* is specified, then only the most recent *count* events are returned.

history keep *[count]*

Changes the size of the history list to *count* events. Initially, 20 events are retained in the history list. If *count* is not specified, the current keep limit is returned.

history nextid

Returns the number of the next event to be recorded in the history list. Use this for printing the event number in command-line prompts.

history redo *[event_number]*

Reruns the command indicated by *event* and returns its result. The default value of *event_number* is **-1**. This command results in history revision. See the following section for details.

History Revision

Pre-8.0 Tcl had a complex history revision mechanism. The current mechanism is more limited, and the **substitute** and **words** history operations have been removed. The **clear** operation was added.

The **redo** history option results in much simpler "history revision." When this option is invoked, the most recent event is modified to eliminate the history command and replace it with the result of the history command. If you want to redo an event without modifying the history, use the **event** operation to retrieve an event, and use the **add** operation to add it to history and execute it.

index_collection

Extracts an object from a collection at a given index and creates a new collection containing only that object. The base collection remains unchanged.

SYNTAX

```
collection index_collection
   coll_A
   index
```

Data Types

coll_A	collection
index	integer

ARGUMENTS

coll_A

Specifies the collection to be searched.

index

Specifies the index into the collection. Allowed values are integers from 0 to one less than the size of the collection.

DESCRIPTION

Use the **index_collection** command to extract a single object from a collection. The result is a new collection containing that object.

The range of indexes is from 0 to one less than the size of the collection. You can find the size of the collection with the **sizeof_collection** command. If the specified index is outside that range, NanoTime issues an error message.

Commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same order each time. Applications that support the sorting of collections allow you to impose a specific order on a collection.

Using the **index_collection** command with the empty collection generates the empty collection as a result and NanoTime issues an error message.

Not all collections can be indexed.

COMMAND PHASING

The **index_collection** command can be executed at any time.

EXAMPLES

The following example shows the first object in a collection being extracted.

```
nt_shell> set c1 [get_cells {u1 u2}]
{"u1", "u2"}
nt_shell> query_objects [index_collection $c1 0]
{"u1"}
```

SEE ALSO

```
collections(2)
query_objects(2)
sizeof_collection(2)
```

is_false

Tests the value of a specified variable, and returns 1 if the value is 0 or the case-insensitive string **false**; returns 0 if the value is 1 or the case-insensitive string **true**.

SYNTAX

```
status is_false
   value
```

Data Types

```
   value      string
```

ARGUMENTS

value

Specifies the name of the variable whose value is to be tested.

DESCRIPTION

This command tests the value of a specified variable, and returns 1 if the value is either 0 or the case-insensitive string **false**. The command returns 0 if the value is either 1 or the case-insensitive string **true**. Any value other than 1, 0, **true**, or **false** generates a Tcl error message.

The **is_false** command is used in writing scripts that test Boolean variables that can be set to 1, 0, or the case-insensitive strings **true** or **false**. When such variables are set to **true** or **false**, they cannot be tested in the negative in an **if** statement by simple variable substitution, because they do not constitute a true or false condition. The following example is not legal Tcl:

```
set x FALSE
if { !$x } {
  set y TRUE
}
```

This results in a Tcl error message, indicating that you cannot use a non-numeric string as the operand of "!". So, although you can test the positive condition, **is_false** allows you to test both conditions safely.

EXAMPLES

The following example shows the use of the **is_false** command:

```
prompt> set x TRUE
TRUE

prompt> if { ![is_false $x] } {
?         set y TRUE
?       }
TRUE

prompt>
```

SEE ALSO

expr(2)
if(2)
is_true(2)

is_true

Tests the value of a specified variable, and returns 1 if the value is 1 or the case-insensitive string **true**; returns 0 if the value is 0 or the case-insensitive string **false**.

SYNTAX

```
status is_true
   value
```

Data Types

```
   value      string
```

ARGUMENTS

value

Specifies the name of the variable whose value is to be tested.

DESCRIPTION

This command tests the value of a specified variable, and returns 1 if the value is either 1 or the case-insensitive string **true**. The command returns 0 if the value is either 0 or the case-insensitive string **false**. Any value other than 1, 0, **true**, or **false** generates a Tcl error message.

The **is_true** command is used in writing scripts that test Boolean variables that can be set to 1, 0, or the case-insensitive strings **true** or **false**. When such variables are set to **true** or **false**, they cannot be tested in the negative in an **if** statement by simple variable substitution, because they do not constitute a true or false condition. The following example is not legal Tcl:

```
set x TRUE
if { !$x } {
  set y FALSE
}
```

This results in a Tcl error message, indicating that you cannot use a non-numeric string as the operand of "!". So, although you can test the positive condition, **is_true** allows you to test both conditions safely.

EXAMPLES

The following example shows the use of the **is_true** command:

```
prompt> set x FALSE
FALSE

prompt> if { ![is_true $x] } {
?         set y FALSE
?       }

FALSE

prompt>
```

SEE ALSO

expr(2)
if(2)
is_false(2)

link_design

Resolves hierarchical references in a design and builds an in-memory representation of the design.

SYNTAX

```
status link_design
   [-keep_capacitive_coupling]
   [-coupling_reduction_factor factor]
   [-disable_lumped_capacitance]
   [-verbose]
   [design_name]
```

Data Types

```
   factor             float
   design_name        string
```

ARGUMENTS

-keep_capacitive_coupling

Causes cross-coupling capacitors to be kept in the RC network data structure, thereby allowing crosstalk analysis to be performed. A NanoTime Ultra license is required. This option cannot be used with the **-coupling_reduction_factor** option and overrides any previous reduction factor setting. If crosstalk analysis is not enabled, all coupling capacitors are split to ground with a factor of 1.0.

-coupling_reduction_factor *factor*

A positive floating-point number that specifies the factor to apply when reducing coupling capacitors to grounded capacitors. The default is 1.0. This option applies to SPICE coupling capacitor elements. It cannot be used with the **-keep_capacitive_coupling** option.

-disable_lumped_capacitance

By default, for nets with only capacitive load and no resistance, the **link_design** command assigns the capacitance to the net's netlist_capacitance attribute. With the **-disable_lumped_capacitance** option, the **link_design** command instead converts these capacitors into parasitic annotations.

-verbose

Causes verbose messages to be displayed during linking.

design_name

Specifies the name of the design to be linked. The default is the current design.

DESCRIPTION

The **link_design** command performs a name-based resolution of hierarchical design references for the specified design or the current design. In addition to resolving references in the design, it builds the internal representation of the design for analysis.

Before you can link a design, you must register the top-level netlist using the **register_netlist** command. You might also need to set the **link_path** and **search_path** variables, depending on where the hierarchical design files are stored. After successfully linking a design, you can proceed with the next analysis step.

To successfully run the **link_design** command, a complete, fully functional design must be connected to all referenced library components and designs. NanoTime finds and links the references to the specified design or current design.

During the link, all files specified in by the **link_path** variable are loaded if they are not already in memory. The goal is a fully instantiated design on which analysis can be performed.

By setting the **link_path** and **search_path** variables, you only need to read the top design and then link it. NanoTime automatically finds and loads other designs and libraries that are needed from the specified paths.

By default, the case sensitivity of the link operation is determined by the source of the objects being linked. You can change this behavior, if necessary, by setting the **link_case** variable. By default, this variable is an empty string, which results in case-sensitive linking without case shifting. Set the variable to **upper** or **lower** to shift all names to uppercase or lowercase.

Unresolved References

If the linker fails to resolve one or more references, first determine and correct the source of the problem, then relink the design using the **link_design** command again. Failures are typically caused by missing libraries or designs, incorrectly specified **link_path** or **search_path** variables, or a file that is in the path but does not have read permission. After making the necessary corrections to your environment, execute the **link_design** command again.

Implicit Reading of SPICE Models

To have **link_design** implicitly read in SPICE transistor models, the SPICE netlist files (specified with the **register_netlist** command) must contain a .lib statement to direct NanoTime to the SPICE model file name. When you link the design with the **link_design** command, NanoTime reads in the SPICE model template into an in-memory technology. If the **link_enable_netlist_spice_model_linking** variable is set to **true** (the default), NanoTime also generates the transistor lookup tables and places them in another in-memory technology during linking. Use the **list_technology** command to view a list of the available technologies.

In the absence of implicit SPICE model reading, you can use the **read_spice_model** command to explicitly read in SPICE format transistor technology information.

Preferred Models

When multiple models exist for the block (for example, a netlist and a timing model) you can use the following variables to specify the preferred models for NanoTime to use. The choice of variable depends 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

Pin, Transistor, and Net Naming Conventions

A number of variables, including the variables in the following list, define the circuit element pin names, transistor names, and power/ground net names that NanoTime recognizes during linking.

```
link_capacitor_term1_pin_name
link_capacitor_term2_pin_name
link_grounded_capacitor_term1_pin_name
link_resistor_term1_pin_name
link_resistor_term2_pin_name
link_transistor_bulk_pin_name
link_transistor_drain_pin_name
link_transistor_gate_pin_name
link_transistor_source_pin_name
link_nmos_alias
link_pmos_alias
link_gnd_alias
link_vdd_alias
```

To view the current settings for these variables, use the command **printvar link***. If you need to change them from their default settings, you must do so before you use the **link_design** command.

Netlist Parasitics

The **link_design** command treats all netlist resistors and capacitors as parasitic elements. It does not represent them as netlist elements, but instead converts them into parasitic annotations. The parasitic annotations come in two forms, lumped net capacitance and RC network.

If the **-disable_lumped_capacitance** option is not used, nets that have purely capacitive loading in the netlist have their capacitance assigned to the netlist_capacitance attribute of the net. This value is also copied into the wire_capacitance_* attribute for net for the **link_design** or **reset_design** command. This capacitance can be overridden with the **set_load** command.

If the net has resistive parasitics, or the **-disable_lumped_capacitance** option is used, then the parasitics are converted into net rc_network annotations, just as in the **read_parasitics** command. These rc_network annotations can be merged with additional extracted parasitics using the **read_parasitics -increment** command, and they might require completion with the **complete_net_parasitics** command to be usable in delay calculation.

Be aware that running the **read_parasitics** command without the **-increment** option, or running the **remove_annotated_parasitics** command, can result in removing parasitics that were created during the **link_design** command. A warning is issued if this might occur. The only way to recover these

parasitics is to run the **link_design** command again.

COMMAND PHASING

The **link_design** command can only be executed in the "startup-complete" state. If issued in a later phase, NanoTime transitions back to the "startup-complete" state and discards all commands and operations performed in the later phase(s) then executes the **link_design** command. After successful execution, the state is changed to "netlist-linked".

EXAMPLES

The following example sets the **search_path** and **link_path** variables, registers three Verilog netlist files, and links the top-level design. NanoTime automatically finds and loads other designs and libraries that are needed from the specified paths.

```
nt_shell> set search_path {. ./../designs}
. ./../designs
nt_shell> set link_path {*}
*
nt_shell> register_netlist -format spice {alu.sp tech.sp}
1
nt_shell> link_design ALU
Compiling "/ ... /designs/alu.sp"
Compiling "/ ... /designs/tech.sp"
Compiling "../libs/tn90lp_v1.0.lib"

Linking design ALU...
Design 'ALU' was successfully linked.

Linking transistor models...
Information: 11032 out of 11032 transistors have all
transistor models linked to them.  (TECH-003)
1
```

SEE ALSO

```
current_design(2)
list_designs(2)
list_libs(2)
register_netlist(2)
read_spice_model(2)
read_parasitics(2)
complete_net_parasitics(2)
remove_annotated_parasitics(2)
set_load(2)
link_capacitor_term1_pin_name(3)
link_enable_netlist_spice_model_linking(3)
link_gnd_alias(3)
link_nmos_alias(3)
link_path(3)
```

```
link_prefer_model(3)
search_path(3)
```

list_attributes

Lists attributes available in NanoTime.

SYNTAX

```
status list_attributes
   [-application]
   [-class class_name]
   [-nosplit]
```

Data Types

```
   class_name      string
```

ARGUMENTS

-application

Lists attributes defined by NanoTime.

-class *class_name*

Limits the listing to attributes of a single class.

-nosplit

Prevents the splitting of lines when columns overflow.

DESCRIPTION

The **list_attributes** command displays an alphabetically sorted list of currently defined attributes.

The **-application** option is required to display attributes. Note that NanoTime does not support user-defined attributes.

There are many application attributes. It is often useful to limit the listing to a specific object class by using the **-class** option. Valid values for the **-class** option are:

capture_window_edge
cell
clock
design
launch_window_edge
lib
lib_cell
lib_pin
lib_topology
model_clock_domain
model_delay_arc
net
parasitic_device
path_arc
pin
port
simulation
timing_check
timing_path
timing_point
topology

To get the value of a single attribute associated with an object, use the **get_attribute** command.

To report the values of all attributes associated with an object, use the **report_attribute** command.

COMMAND PHASING

The **list_attributes** command is not phase-restricted.

EXAMPLES

The following command lists clock attributes only:

```
nt_shell> list_attributes -application -class clock
...

Properties:
    A - Application-defined
    U - User-defined
    I - Importable from design/library

Attribute Name              Object    Type     Properties  Constraints
-------------------------------------------------------------------------

clock_latency_fall_max      clock     float    A
clock_latency_fall_min      clock     float    A
clock_latency_rise_max      clock     float    A
clock_latency_rise_min      clock     float    A
...
```

SEE ALSO

```
get_attribute(2)
report_attribute(2)
```

list_designs

Lists designs that have been read into NanoTime.

SYNTAX

```
status list_designs
   [-all]
   [-only_used]
```

ARGUMENTS

-all

Lists all designs, including those that are instantiated in the current design but are not in memory. In the absence of this option, only the designs in memory are listed.

-only_used

Lists only designs that are currently being used. This includes the current design, any design that is linked or partially linked, and any designs instantiated in a linked design.

DESCRIPTION

The **list_designs** command lists the designs that have been read into NanoTime. By default, only designs that are currently in memory are listed. Using the **-all** option lists the designs that are instantiated in a linked design but are not in memory.

You can prevent the listing of unused designs by using the **-only_used** option.

The list generated by the command shows the status of the design using the following notation:

* - Indicates that the design is the current design L - Indicates that the design is linked. N - Indicates that the design is not in memory. l - Indicates that the design is partially linked.

COMMAND PHASING

The **list_designs** command is not phase-restricted.

EXAMPLES

The following command reports the designs in memory.

```
nt_shell> list_designs

Design Registry:
 *L AD4FULA        /u/abc/designs/top.db:AD4FULA
    AD4PG          /u/abc/designs/add1.db:AD4PG
    ADD5A          /u/abc/designs/add2.db:ADD5A
    MULT1          /u/abc/designs/top.db:MULT1
```

The following command reports only the designs that are in use and in memory.

```
nt_shell> list_designs -only_used

Design Registry:
 *L AD4FULA        /u/abc/designs/top.db:AD4FULA
    ADD5A          /u/abc/designs/add2.db:ADD5A
    MULT1          /u/abc/designs/top.db:MULT1
```

SEE ALSO

current_design(2)
link_design(2)
register_netlist(2)

list_lib_topology

Displays a list of lib_topology objects.

SYNTAX

```
status list_lib_topology
   [-library name]
   [-only_used]
   [-output filename]
```

Data Types

```
   name              string
   filename          string
```

ARGUMENTS

-library *name*

Restricts the listing to lib_topology objects contained in the specified library.

-only_used

Lists only the lib_topology objects found to exist in the design, based on the setting of the lib_topology **matched_count** attribute.

-output *filename*

Writes the output of the command to a file rather than to the screen.

DESCRIPTION

The **list_lib_topology** command displays a list of lib_topology objects. The listing includes lib_topology objects in the common library, in the global library, and in libraries that have been read with the **read_topology_library** command. The listing shows the name (in the format *library_name.topology_name*), description string, and version string of each lib_topology object.

To restrict the listing to only the topologies contained in a specific library, use the **-library** option.

To display a list of only those library topologies that exist in the design, use the **list_lib_topology -only_used** command. The command lists only those lib_topology objects that have the **matched_count** attribute set to a value greater than zero. The **matched_count** attribute is set by the **match_topology**, **check_topology**, or **set_search_enabled -only_used** commands.

COMMAND PHASING

The **list_lib_topology** command can be executed in at any time after the **link_design** command.

EXAMPLES

The following command lists all lib_topology objects.

```
nt_shell> list_lib_topology
Lib Topology Registry

Name                    Description        Version
----------------------- ------------------ ---------
common.bignand_latch    bignand_latch      V20061218
common.bignand_latch2   bignand_latch2     V20061218
 ...
```

The following command lists the lib_topology objects in the user library user2.

```
nt_shell> list_lib_topology -library user2
 ...
```

The following commands list the lib_topology objects that exist in the design.

```
nt_shell> set_search_enabled -only_used
1
nt_shell> list_lib_topology -only_used
 ...
```

SEE ALSO

```
list_topology_library(2)
match_topology(2)
read_topology_library(2)
set_search_enabled(2)
hierarchy_separator(3)
```

list_libs

Lists all .db libraries that have been read into NanoTime with the **read_library** command.

SYNTAX

```
status list_libs
   [-only_used]
```

ARGUMENTS

-only_used

Lists only the libraries that are currently being used. A library is in use if a linked design has links to cells in that library.

DESCRIPTION

The **list_libs** command lists the libraries that are read into NanoTime. You can filter unused libraries out of the display by using the **-only_used** option. A library is in use if a linked design links to library cells from the library.

COMMAND PHASING

The **list_libs** command is not phase-restricted.

EXAMPLES

The following example lists all libraries.

```
nt_shell> list_libs
```

```
    Library Registry:
        bus1_lib      /u/abc/lib/bus1_lib.db:bus1_lib
        tech1         /u/abc/lib/tech1.db:tech1
```

SEE ALSO

current_design(2)
link_design(2)
list_designs(2)

list_net_groups

Lists the nets assigned to groups by the **create_net_group** command.

SYNTAX

```
status list_net_groups
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **list_net_groups** command lists the net groups and the nets assigned to each group. Nets are assigned to groups by the **create_net_group** command

COMMAND PHASING

The **list_net_groups** command is not phase-restricted.

EXAMPLES

The following command lists the net groups and the nets belonging to each group.

```
nt_shell> list_net_groups
 ...
Group Name    Nets
------------ --------------------------------
SBUS          {SBUS[0] SBUS[1] SBUS[2] SBUS[3]}
xor           {xor xnor xor0 xor1 xor2 xor3}
```

SEE ALSO

```
create_net_group(2)
remove_net_group(2)
```

list_netlists

Lists the netlist files that have been registered with the **register_netlist** command.

SYNTAX

```
status list_netlists
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **list_netlists** command displays a list of netlists currently in the netlist registry. These are the netlist files that have been registered with the **register_netlist** command. For each netlist file, the list shows the linking status, the file type, and the full file name.

COMMAND PHASING

The **list_netlists** command is not phase-restricted.

EXAMPLES

The following command lists the files that have been registered with the **register_netlist** command. The notation *L in the first column of the report indicates that the netlist has been linked.

```
nt_shell> list_netlists
Netlist Registry:

     Format     File File Name
```

```
-------------------------------------------------
 *L  spice      /remote/t1/user1/mydir/alu.sp
 *L  spice      /remote/t1/user1/mydir/tech.sp
1
```

SEE ALSO

```
link_design(2)
register_netlist(2)
remove_netlist(2)
```

list_patterns

Lists the patterns stored in the pattern registry by the **read_pattern** command.

SYNTAX

```
status list_patterns
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **list_patterns** command lists the netlist patterns currently stored in the pattern registry. Patterns can be added to the registry with the **read_pattern** command and removed with the **remove_pattern** command.

COMMAND PHASING

The **list_patterns** command is not phase-restricted.

EXAMPLES

The following example lists the patterns that are stored in the pattern registry.

```
nt_shell> list_patterns
Pattern Registry

Pattern Name
----------------------------------------
```

```
pre_an2k
1
```

SEE ALSO

```
foreach_match(2)
read_pattern(2)
remove_pattern(2)
```

list_technology

Lists information about the technology data currently loaded into memory and available for linking in NanoTime.

SYNTAX

```
status list_technology
   [-only_used]
```

ARGUMENTS

-only_used

Lists only the technology registry entries that are currently linked to the design.

DESCRIPTION

The **list_technology** command lists the technology libraries that have been read into NanoTime. Here is an example of a technology listing:

```
nt_shell> list_technology

Technology Registry

Attributes (T):
    t - Tech data from techfile
    m - Tech data from SPICE model
    s - SPICE model

Name     Voltage    Temp Nmos    Pmos    T Source file
-------- ------- ------- ------- ------- - ------------
typ        1.000  85.000 nch     pch     m <netlist>
typ        1.000  85.000 nch     pch     s <netlist>
```

For each technology, the report shows the technology name, supply voltage, temperature, NMOS transistor model name, PMOS transistor model name, technology attributes, and model source file.

The "Attributes" column uses the letter codes t, m, or s to indicate the type of technology.

The letter t indicates a set of lookup models taken from a techfile (read with the **read_techfile** command).

The letter m indicates a set of lookup models generated by NanoTime from a SPICE model template. This model generation can occur with the **link_design**, **set_technology**, or **check_design** commands, depending on the analysis flow.

The letter s indicates a SPICE model template read with the **read_spice_model** command, or read implicitly during the **link_design** command phase with the **link_enable_netlist_spice_model_linking** variable set to **true**.

The "Source file" column indicates the name of the file from which the technology was obtained, which could be a techfile or a SPICE model file. The notation "<netlist>" indicates that the SPICE models were read in implicitly during the **link_design** command phase.

For additional reporting of technology information, use the **report_technology** command.

COMMAND PHASING

The **list_technology** command is not phase-restricted.

EXAMPLES

The following example lists all the technology files that are currently being used.

```
nt_shell> list_technology -only_used

Technology Registry

Attributes (T):
    t - Tech data from techfile
    m - Tech data from SPICE model
    s - SPICE model

Name    Voltage    Temp Nmos   Pmos    T Source file
------- ------- ------- ------ ------- - -----------
typ       1.000  85.000 nch    pch     m <netlist>
```

SEE ALSO

```
check_design(2)
link_design(2)
read_spice_model(2)
read_techfile(2)
register_netlist(2)
report_technology(2)
```

```
set_technology(2)
link_enable_netlist_spice_model_linking(3)
```

list_topology_library

Lists the topology libraries.

SYNTAX

```
status list_topology_library
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **list_topology_library** command lists the topology libraries, including the common library, global library, and any user topology libraries that have been read with the **read_topology_library** command. The list shows the topology library name, its description string, and its version number string.

COMMAND PHASING

The **list_topology_library** command can be executed at any time after the **link_design** command.

EXAMPLES

The following lists the topology libraries:

```
nt_shell> list_topology_library
Topology Library Registry

Name      Description                    Version
```

```
--------- ----------------------------------------
common    Builtin common pattern library  V20061218
global    Global pattern library          V20070126
user2     User pattern library            V20070126
```

SEE ALSO

```
get_lib_topology(2)
list_lib_topology(2)
read_topology_library(2)
```

lminus

Removes one or more named elements from a list and returns a new list.

SYNTAX

```
list lminus
   [-exact]
   original_list
   elements
```

Data Types

```
   original_list     list
   elements          list
```

ARGUMENTS

-exact

Specifies that the exact pattern is to be matched. By default, **lminus** uses the default match mode of **lsearch**, the **-glob** mode.

original_list

Specifies the list to copy and modify.

elements

Specifies a list of elements to remove from *original_list*.

DESCRIPTION

The **lminus** command removes elements from a list by using the element itself, rather than the index of the element in the list (as in **lreplace**). The **lminus** command uses the **lsearch** and **lreplace** commands to find the elements and replace them with nothing.

If none of the elements are found, a copy of *original_list* is returned.

The **lminus** command is often used in the translation of Design Compiler scripts that use the subtraction operator (-) to remove elements from a list.

EXAMPLES

The following example shows the use of the **lminus** command. Notice that no error message is issued if a specified element is not in the list.

```
prompt> set l1 {a b c}
a b c

prompt> set l2 [lminus $l1 {a b d}]
c

prompt> set l3 [lminus $l1 d]
a b c
```

The following example illustrates the use of **lminus** with the **-exact** option:

```
prompt> set l1 {a a[1] a* b[1] b c}
a a[1] a* b[1] b c

prompt> set l2 [lminus $l1 a*]
{b[1]} b c

prompt> set l3 [lminus -exact $l1 a*]
a {a[1]} {b[1]} b c

prompt> set l4 [lminus -exact $l1 {a[1] b[1]} ]
a a* b c
```

SEE ALSO

```
lreplace(2)
lsearch(2)
```

ls

Lists the contents of a directory.

SYNTAX

```
string ls [filename ...]

string filename
```

ARGUMENTS

filename

Provides the name of a directory or filename, or a pattern which matches files or directories.

DESCRIPTION

If no argument is specified, the contents of the current directory are listed. For each *filename* matching a directory, **ls** lists the contents of that directory. If *filename* matches a file name, the file name is listed.

EXAMPLES

```
shell> ls *.db *.pt

test1.pt         c1.db          c3.db          c5.db

test2.pt         c2.db          c4.db          c6.db
```

SEE ALSO

cd(2)
pwd(2)

man

Displays reference manual pages.

SYNTAX

```
string man
   topic
```

Data Types

```
   topic      string
```

ARGUMENTS

topic

Specifies the subject to display. Available topics include commands, variables, and error messages

DESCRIPTION

The **man** command displays the online manual page for a command, variable, or error message. You can write man pages for your own Tcl procedures and access them with the **man** command by setting the **sh_user_man_path** variable to an appropriate value. See the man page for the **sh_user_man_path** variable for details.

EXAMPLES

The following command displays the man page for the **echo** command:

```
prompt> man echo
```

The following command displays the man page for the error message CMD-025:

```
prompt> man CMD-025
```

SEE ALSO

```
help(2)
sh_user_man_path(3)
```

mark_clock_gate

Identifies a clock gate structure in the design.

SYNTAX

```
collection mark_clock_gate
   -clock pin_list
   -output net
   [-positive_enable pin_list]
   [-negative_enable pin_list]
   [-min_fall_controlling_pin pin]
   [-min_rise_controlling_pin pin]
   [-max_fall_controlling_pin pin]
   [-max_rise_controlling_pin pin]
   [-transparent]
   [-name structure_name]
```

Data Types

```
   pin                string
   pin_list           list
   net                string
   structure_name     string
```

ARGUMENTS

-clock *pin_list*

Identifies the clock pin of the clock gate (an input to the clock gate).

-output *net*

Identifies the output net (the gated clock signal) of the clock gate.

-positive_enable *pin_list*

Identifies the positive enable pins of the clock gate. All of the listed pins must be logic 1 to enable the clock on the output net.

-negative_enable *pin_list*

Identifies the negative enable pins of the clock gate. All of the listed pins must be logic 0 to enable the

clock on the output net.

-min_fall_controlling_pin ***pin***

Identifies the clock pin controlling the output falling during min path tracing through the clock gate. All options **-min_fall_controlling_pin**, **-min_rise_controlling_pin**, **-max_fall_controlling_pin** and **-max_rise_controlling_pin** must be defined if any of them are defined.

-min_rise_controlling_pin ***pin***

Identifies the clock pin controlling the output rising during min path tracing through the clock gate.

-max_fall_controlling_pin ***pin***

Identifies the clock pin controlling the output falling during max path tracing through the clock gate.

-max_rise_controlling_pin ***pin***

Identifies the clock pin controlling the output rising during max path tracing through the clock gate.

-transparent

Allows transparent checking of the clock gate.

-name ***structure_name***

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes clock gating structures in the design, such as a clock signal entering a NAND or NOR gate. NanoTime automatically defines timing checks on these structures to ensure that the enable signals arrive before the applicable clock edges.

The **mark_clock_gate** command lets you specify the location of a clock gating structure that NanoTime does not recognize automatically. In the command, you must specify the input clock pin and the output (gated clock) net. In addition, specifying the positive or negative enable pin or pins of the clock gate helps NanoTime to determine when the gated clock signal is enabled. If using the options to control path traversal through the clock gate with -min/max_rise/fall_controlling_pin options then all the **-min_fall_controlling_pin**, **-min_rise_controlling_pin**, **-max_fall_controlling_pin** and **-max_rise_controlling_pin** options must be defined.

To cancel the effects of the **mark_clock_gate** command, or to override recognition of an automatically detected clock gate, use the **erase_clock_gate** command.

COMMAND PHASING

The **mark_clock_gate** command is typically executed between the **link_design** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

You might need to ensure that a clock propagation step occurs after manually marking clock gates. For this reason, you might want to mark the clock gates before the first **match_topology** command. In addition, if the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. Clock propagation does not occur at the **check_topology** command unless there is no **match_topology** command in the flow.

If the **mark_clock_gate** command is used in a lib_topology definition, it should be in the pre_match_topology phase point.

EXAMPLES

The first following command identifies a clock gate structure in the design having a clock pin named clk, an output net named out1, and positive enable pins named EN1 and EN2. When both EN1 and EN2 are logic 1, the clock signal passes through the structure from the clock pin clk to the output net out1.

```
nt_shell> mark_clock_gate -clock clk \
          -output out1 -positive_enable { EN1 EN2 }
```

```
The second following command identifies a clock gate structure where there are no enables, but wher
there are four clock pins, X1.MN0.g, X1.MN1.g, X1.MP0.g, and X1.MP1.g which are the min_fall, max_f
min_rise, and max_rise controlling pins, respectively.
nt_shell> mark_clock_gate -clock {X1.MN0.g X1.MN1.g X1.MP0.g X1.MP1.g} \
      -min_fall_controlling_pin X1.MN0.g -max_fall_controlling_pin X1.MN1.g \
      -min_rise_controlling_pin X1.MP0.g -max_rise_controlling_pin X1.MP1.g \
      -output out1
```

SEE ALSO

```
create_lib_topology(2)
erase_clock_gate(2)
topo_auto_search_class(3)
match_topology(2)
topo_match_topology_reset_clock_and_topology(3)
topo_clock_gate_allow_reconvergent_clocks(3)
topo_clock_gate_depth(3)
topo_clock_gate_strict_checking(3)
```

mark_clock_network

Forces or prevents clock propagation through specified ports, pins, and nets.

SYNTAX

```
collection mark_clock_network
   -force_propagation | -stop_propagation
   [-no_pulse]
   [-add_setup]
   [-add_hold]
   [-transparent]
   [-end_dcs]
   object_list
```

Data Types

```
   object_list          list
```

ARGUMENTS

-force_propagation

Forces clock propagation through the specified objects.

-stop_propagation

Prevents clock propagation through the specified objects.

-no_pulse

Prevents the pulse clock property from being applied at this net and the remaining clock tree driven by this net. This prevents pulse clock related cycle checking/phasing rules from being applied and reverts back to applying non-pulse cycle checking/phasing rules when evaluating timing checks. Use this option for pulse clocks with large duty cycles as indicated by the TIMI-041 message.

-add_setup

Adds setup checks at the stopped clock when the **-stop_propagation** option is used.

-add_hold

Adds hold checks at the stopped clock when the **-stop_propagation** option is used.

-transparent

Makes the added setup or hold checks transparent, which allows the non-clock signal intersecting with the clock to continue being propagated if it is within the transparent window of the clock pulse.

-end_dcs

Makes a dynamic clock simulation region terminate at the specified objects.

object_list

Specifies the list of ports, pins, or nets affected by the command. If the list is given as the names of objects, the names will be first searched as the ports of the top-level design followed by pins and nets. But when **mark_clock_network** command is issued within **foreach_match** command, NanoTime will not search for the ports of the top-level design.

DESCRIPTION

Correct clock propagation is necessary for recognizing sequential elements and performing timing checks. By default, NanoTime propagates clocks through inverters, through the source/drain pathway of transistors that are turned on, through recognized clock gating structures, and through static logic when all control inputs allow it. Clock propagation stops when a clock reaches the clock node of a latch, a primary output specified with the **set_output_delay** command, or a domino logic precharge node.

If NanoTime does not correctly propagate clocks by default, you can use the **mark_clock_network** command to force specific ports, pins, and nets to either propagate or not propagate clocks. In the command, use either the **-force_propagation** or **-stop_propagation** option and specify a list of objects that are to propagate or not propagate clocks.

To stop clock propagation at a specified point in the clock network, use the **-stop_propagation** option. Clock propagation stops at the marked object. Any clock paths traceable beyond this point are not considered part of the clock network. No timing checks are performed for sequential devices beyond that point.

By default, no timing checks are done at a stop-propagation point. If that point is a data and clock convergence point where timing checks should be done, specify the desired types of timing checks with the **-add_setup** and **-add_hold** options. To have these timing checks treated as transparent, use the **-transparent** option as well, which allows the non-clock signal intersecting with the clock to continue being propagated if it is within the transparent window of the clock pulse.

To cancel the effects of the **mark_clock_network** command, use the **erase_clock_network** command.

If the **mark_clock_network** command is used in a lib_topology definition, it should be in the pre_match_topology phase point.

COMMAND PHASING

The **mark_clock_network** command is typically used after the **link_design** command and before the **match_topology** command.

You might need to ensure that a clock propagation step occurs after manually marking the clock network. For this reason, you might want to use this command before the first **match_topology** command. In addition, if the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command. Clock propagation does not occur at the **check_topology** command unless there is no **match_topology** command in the flow.

EXAMPLES

The following command forces clock propagation through any ports, pins, or nets named N1 or N2.

```
nt_shell> mark_clock_network \
          -force_propagation { N1 N2 }
```

The following command prevents clock propagation through any ports, pins, or nets named N3 or N4.

```
nt_shell> mark_clock_network \
          -stop_propagation { N3 N4 }
```

SEE ALSO

create_lib_topology(2)
erase_clock_network(2)
match_topology(2)
topo_match_topology_reset_clock_and_topology(3)

mark_correlated_region

Identifies where correlated arrivals are needed for more accurate delay calculation.

SYNTAX

```
status mark_correlated_region
   -inputs ports_pins_or_nets
   -outputs ports_or_nets
```

Data Types

```
   ports_pins_or_nets      list
   ports_or_nets           list
```

ARGUMENTS

-inputs *ports_pins_or_nets*

The starting ports, pins or nets of the correlated region.

-outputs *ports_or_nets*

The ports or nets where correlated arrivals occur.

DESCRIPTION

When calculating the delay for a single stage of a path, NanoTime by default uses a single trigger event and sets all other side inputs to fixed logic states. In some situations, one or more of these side inputs are logically and temporally correlated to the trigger transition and should also be analyzed to calculate a more accurate delay. To use multiple switching inputs during a single delay calculation it is necessary to specify these inputs and compute the delay offset between them. The **mark_correlated_region** command lets you specify the relationships necessary for determining the offsets between correlated pins.

The **-inputs** option specifies a list of ports, pins or nets that are the common starting points for timing paths to a set of correlated nets. The **-outputs** option specifies a list of ports or nets that are correlated

points where skews need to be determined. The output points are restricted to nets driving select pins of mux topologies. Every output must be in the transitive fanout of one or more of the inputs specified. The intersection of the transitive fanout of the inputs and the transitive fanin of the outputs defines a correlated region. A pin can only be in one correlated region.

When these regions have been determined, the path tracer computes the best and worst case paths to each output from a single transition on an input. When using these offsets for delay calculation, if the skew between a trigger and another correlated pin is negative then it is ignored. Only positive offsets, i.e. signals arriving later than the trigger, are used at correlated pins.

The Boolean pin attribute **is_correlated_output** has the value of **true** if the pin is part of a correlated set. The pin attribute **correlated_output_pins** contains a collection of correlated pins.

Alternatively, you can use the **set_correlated_input** command instead to manually specify the skew between correlated inputs.

COMMAND PHASING

The **mark_correlated_region** command can be executed any time after the **link_design** command but before the **check_design** command.

EXAMPLES

The following command specifies a correlated region that starts from nets A and B, and has correlated arrivals at the nets S0, S1, S2, and S3.

```
nt_shell> mark_correlated_region \
          -inputs {A B} \
          -outputs {S0 S1 S2 S3}
```

SEE ALSO

set_correlated_input(2)

mark_cross_coupled_pmos

Identifies a cross-coupled PMOS (pullup) structure in the design.

SYNTAX

```
collection mark_cross_coupled_pmos
   -transistors devices
   [-name structure_name]
```

Data Types

```
   devices            list
   structure_name     string
```

ARGUMENTS

-transistors *devices*

Identifies the two transistor instances that form a cross-coupled PMOS element.

-name *structure_name*

Assigns a structure name to the topology, which can be used with the **get_topology** command.

DESCRIPTION

This command identifies cross-coupled PMOS transistors which typically pull up two output nets. The devices must be PMOS devices and cross-coupled. Typically, this circuit structure is used in low power differential output structures. NanoTime does not support automatic recognition of this topology.

NanoTime issues an error message if the structure does not conform to a conventional cross-coupled PMOS topology.

To cancel the effects of this command, use the **erase_cross_coupled_pmos** command.

COMMAND PHASING

The **mark_cross_coupled_pmos** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command marks transistor instances MP1 and MP2 as a cross-coupled PMOS transistor structure.

```
nt_shell> mark_cross_coupled_pmos -transistors {MP1 MP2}
```

SEE ALSO

erase_cross_coupled_pmos(2)
report_topology(2)

mark_differential_synchronizer

Identifies a differential synchronizer structure in the design.

SYNTAX

```
status mark_differential_synchronizer
   -transistors elements
   [-tapped_transistors tapped_elements]
   [-name structure_name]
```

Data Types

```
   elements           list
   tapped_elements    list
   structure_name     string
```

ARGUMENTS

-transistors *elements*

Identifies the transistor instances in the list as part of a differential synchronizer.

-tapped_transistors *tapped_elements*

Identifies the transistor instances in the list as needed to drive the tapped outputs of the differential synchronizer.

-name *structure_name*

Assigns a structure name that can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes differential synchronizer circuits between two nets that form a differential net pair if the **topo_auto_search_class** variable includes the **differential_synchronizer** keyword.

If NanoTime does not recognize a differential synchronizer, you can manually mark differential synchronizers with the **mark_differential_synchronizer** command. The transistor instances named in the command are marked as part of the differential synchronizer. Note that all the transistors of the differential synchronizer must be specified in this command.

Before you use the **mark_differential_synchronizer** command, you must mark all differential net pairs associated with the differential synchronizer by using the **set_differential** command. This includes the enclosing nets and any differential clock signals that feed into the transistors in the synchronizer.

All transistors in a differential synchronizer, whether they are automatically recognized or manually marked, are inferred to be feedback transistors except those that are identified as tapped transistors. This prevents unnecessary timing arcs inside or through the synchronizer structure.

The transistors in a differential synchronizer together must form a loop consisting of two channel-connected blocks (CCBs). Each of the CCBs must drive from one enclosing differential net to the other. Each CCB can also drive a tapped output net. Marking transistors as tapped allows path tracing through those transistors. The tapped outputs must form a differential net pair which should be marked with the **set_differential** command before executing the **mark_differential_synchronizer** command.

Timing paths across the synchronizer (between the enclosing nets) are supported if all of the synchronizer transistors are also tapped transistors.

If a differential synchronizer exists between the input and output nets of an inverter, the nets must be marked as forming a differential pair and the synchronizer must be manually marked.

NanoTime issues a TLIB-005 warning if one of the transistor instances specified in the **mark_differential_synchronizer** command belongs to an existing differential synchronizer. NanoTime issues a TOPO-092 error message if it cannot find a differential net pair associated with the specified differential synchronizer.

To cancel the effects of this command, use the **erase_differential_synchronizer** command.

COMMAND PHASING

The **mark_differential_synchronizer** command must be used after the **link_design** command but before the **match_topology** command.

EXAMPLES

The following command marks a differential synchronizer on transistor instances MP1, MN1, MP2, and MN2:

```
nt_shell> mark_differential_synchronizer -transistors "MP1 MN1 MP2 MN2"
```

SEE ALSO

```
erase_differential_synchronizer(2)
report_topology(2)
set_differential(2)
topo_auto_search_class(3)
```

mark_feedback

Identifies a feedback structure in the design.

SYNTAX

```
collection mark_feedback
   -transistors elements
   [-name structure_name]
```

Data Types

```
   elements             list
   structure_name       string
```

ARGUMENTS

-transistors *elements*

A list of transistor instances to be marked as feedback devices.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime recognizes feedback transistors connected between the input and output of an inverter. The tool removes them from path searches to prevent looping through the circuit. However, feedback devices are considered to be active devices during delay calculation.

Automatic recognition of feedback transistors is based on the connectivity and resistance of each transistor. In cases where automatic recognition does not work as expected due to resistance considerations, you can change the behavior by modifying the values of the **topo_feedback_inv_ratio**, **topo_feedback_p_res_ratio**, and **topo_feedback_n_res_ratio** variables.

Alternatively, you can manually mark transistor instances as feedback devices by using the **mark_feedback** command.

To cancel the effects of this command, use the **erase_feedback** command.

COMMAND PHASING

The **mark_feedback** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you use the **mark_feedback** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

If you use the **mark_feedback** command in a lib_topology definition, it should be in the pre_match_topology phase point.

EXAMPLES

The following command marks transistor instance MP1 as a feedback transistor:

```
nt_shell> mark_feedback -transistors MP1
```

SEE ALSO

```
create_lib_topology(2)
erase_feedback(2)
topo_feedback_inv_ratio(3)
topo_feedback_p_res_ratio(3)
topo_feedback_n_res_ratio(3)
```

mark_flip_flop

Identifies a flip-flop structure in the design.

SYNTAX

```
collection mark_flip_flop
   -master_latch master_object
   -slave_latch slave_object
   [-name structure_name]
```

Data Types

```
   master_object          string
   slave_object           string
   structure_name         string
```

ARGUMENTS

-master_latch *master_object*

Identifies the latch topology object or the latch net of the master latch.

-slave_latch *slave_object*

Identifies the latch topology object or the latch net of the slave latch.

-name *structure_name*

The structure name assigned to the topology.

DESCRIPTION

NanoTime can automatically recognize some flip-flop structures in the design and define timing checks for them. However, flip-flop recognition is not enabled by default. To enable automatic recognition, add the term **flip_flop** to the **topo_auto_search_class** variable.

The **mark_flip_flop** command lets you specify a flip-flop topology that NanoTime does not recognize automatically.

A flip-flop is a structure where two latches are connected together sequentially with both latches clocked by the same clock, but with an inverted phase relationship. The first latch is the master, which must have a transparent path to the second latch. The second latch is the slave, which is evaluated as a nontransparent device. The clock that drives the slave must be an inverted version of the clock that drives the master.

In the **mark_flip_flop** command, you must specify the master object and slave object of the flip-flop. For each object, you can specify either the latch net or the latch topology object previously recognized by a **match_topology** or **check_topology** command, or marked manually with the **mark_latch** command. To get the latch topology object, use the **get_topology** command.

For accurate flip-flop recognition, the constituent latches must already be recognized or marked. To accomplish this, you might need to execute a **match_topology** command before manually marking flip-flops.

By default, NanoTime uses the latch net as the path endpoint for setup and hold checks. You can change the places where NanoTime performs timing checks by using the **set_timing_check_attachment** command.

To cancel the effects of the **mark_flip_flop** command, or to override recognition of an automatically detected flip-flop, use the **erase_flip_flop** command.

COMMAND PHASING

The **mark_flip_flop** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you use the **mark_flip_flop** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

If you modify the clock network or manually mark latches after the first **match_topology** command, you might need to force another clock propagation operation before trying to recognize or mark flip-flops. To do this, set the **topo_match_topology_reset_clock_and_topology** variable to **true** and execute the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

If you use the **mark_flip_flop** command in a lib_topology definition, it should be in the post_match_topology phase point.

EXAMPLES

The following command specifies a flip-flop that has its master latch node at net mn1 and its slave latch node at sn1.

```
nt_shell> mark_flip_flop \
          -master_latch [get_nets mn1] \
          -slave_latch [get_nets sn1]
```

SEE ALSO

erase_flip_flop(2)
get_topology(2)
mark_latch(2)
match_topology(2)
set_timing_check_attachment(2)
topo_auto_search_class(3)
topo_flip_flop_strict_slave_check(3)
topo_match_topology_reset_clock_and_topology(3)

mark_instance

Marks an instance in the design with path tracing instructions.

SYNTAX

```
status mark_instance
   [-dont_search_thru_gate]
   [-dont_search_thru_channel]
   [-dont_include_for_side_branch]
   [-exclude_from_dcs]
   [-exclude_from_topology]
   elements
```

Data Types

```
   elements       list
```

ARGUMENTS

-dont_search_thru_gate

Prevents path tracing through the gate pin of the transistor.

-dont_search_thru_channel

Prevents path tracing through the source or drain pin of the transistor.

-dont_include_for_side_branch

Prevents the transistor from being included as a side branch. It turns off the side branch function on the specified transistor. A side branch of a stage is a transistor that does not affect the output transition. Turning the transistor on or off changes the loading and can affect the delay.

-exclude_from_dcs

Prevents the transistor's channel-connected block as well as its transitive fanout from being included in a dynamic clock simulation (DCS) region (if DCS is enabled and the transistor normally belongs in the DCS region).

-exclude_from_topology

Causes the transistor to be not considered for clock propagation and automatic latch topology

recognition, but considered for timing and noise simulations.

elements

The list of transistor elements in the design that are to be marked with the path tracing settings.

DESCRIPTION

The **mark_instance** command provides special path tracing instructions for specified transistors in the design.

The command can be used multiple times to set different options on the same transistor.

Excluding a transistor from a path tracing operation also prevents path tracing and noise analysis from occurring on all paths downstream from that transistor.

To cancel the effects of the **mark_instance** command, use the **erase_instance** command.

COMMAND PHASING

The **mark_instance** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command marks transistor instance m1 so that a path search does not go through the gate of the transistor.

```
nt_shell> mark_instance -dont_search_thru_gate m1
```

SEE ALSO

```
erase_instance(2)
trace_paths(2)
```

mark_inverter

Identifies an inverter in the design.

SYNTAX

```
collection mark_inverter
   -n_transistor elements
   -p_transistor elements
   [-name structure_name]
```

Data Types

```
   elements             list
   structure_name       string
```

ARGUMENTS

-n_transistor *elements*

Identifies the pulldown transistors of the inverter.

-p_transistor *elements*

Identifies the pullup transistors of the inverter.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes the standard CMOS inverter structure consisting of one PMOS transistor and one NMOS transistor. NanoTime also recognizes the inverter of channel connected PMOS transistors and NMOS transistors. NanoTime uses inverter information to propagate clocks and to trace data signals through the design.

The **mark_inverter** command lets you specify the location of an inverter that NanoTime does not recognize automatically. In the command, you specify the NMOS pulldown transistors and the PMOS pullup transistors of the inverter.

To cancel the effects of the **mark_inverter** command, or to override recognition of an automatically detected inverter, use the **erase_inverter** command.

COMMAND PHASING

The **mark_inverter** command is typically executed between the **match_topology** command and the **check_topology** command. You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command identifies an inverter having an NMOS pulldown transistor N11 and a PMOS pullup transistor P18.

```
nt_shell> mark_inverter -n_transistor N11 \
          -p_transistor P18
```

SEE ALSO

erase_inverter(2)
match_topology(2)

mark_latch

Identifies a latch structure in the design.

SYNTAX

```
collection mark_latch
   -latch_net net
   [-inputs pins_or_nets |
     -rise_inputs pins_or_nets |
     -fall_inputs pins_or_nets]
   [-output latch_out]
   [-feedback objects]
   [-tapped_feedback objects]
   [-feedforward objects]
   [-clock pin_list |
     -io_ff_clock pin_list |
     -io_fr_clock pin_list |
     -io_rf_clock pin_list |
     -io_rr_clock pin_list]
   [-setup_to location]
   [-hold_to location]
   [-name structure_name]
   [-match_pattern]
```

Data Types

net	string
pins_or_nets	list
latch_out	string
objects	list
pin_list	list
location	string
structure_name	string

ARGUMENTS

-latch_net *net*

The latch node (the net that holds the latch value).

-inputs *pins_or_nets*

The argument is a list of the input pins or nets of the latch. This option is mutually exclusive with the **-rise_inputs** and **-fall_inputs** options.

-rise_inputs ***pins_or_nets***

The argument is a list of the pins or nets that cause the latch node to rise. This is usually the gate pin or net of a PMOS pull-up transistor. This option is mutually exclusive with the **-inputs** option and must be used with the **-fall_inputs** option.

-fall_inputs ***pins_or_nets***

The argument is a list of the pins or nets that cause the latch node to fall. This is usually the gate pin or net of an NMOS pull-down transistor. This option is mutually exclusive with the **-inputs** option and must be used with the **-rise_inputs** option.

-output ***latch_out***

The output pin or net that indicates the latch value.

-feedback ***objects***

The latch elements that feed the value from the latch output back to the latch node.

-tapped_feedback ***objects***

The latch elements that have already been marked as feedback which also drive signals outside the latch through a non-feedback path.

-feedforward ***objects***

The elements that feed the value from the latch node to the feedback elements.

-clock ***pin_list***

The clock pin of the latch. If you do not specify a clock pin, the **check_topology** command tries to find one. If NanoTime does not find a clock pin for a latch, a **check_topology** error results. You must correct the error before you can execute the **check_design** command.

-io_ff_clock ***pin_list***

The clock pins that capture the input falling to latch value falling transitions.

-io_fr_clock ***pin_list***

The clock pins that capture the input falling to latch value rising transitions.

-io_rf_clock ***pin_list***

The clock pins that capture the input rising to latch value falling transitions.

-io_rr_clock ***pin_list***

The clock pins that capture the input rising to latch value rising transitions.

-setup_to ***location***

The place in the latch structure where setup checking is to be performed. The allowable values are **latch_net** (the default), **output**, or **input**.

-hold_to ***location***

The place in the latch structure where hold checking is to be performed. The allowable values are **latch_net** (the default) or **input**.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

-match_pattern

Instructs NanoTime to build a latch pattern with the nets and devices specified in the command and mark all latches matching the same pattern.

DESCRIPTION

NanoTime automatically recognizes latch structures of various types, based on the meeting of clock and data signals where inverter, NAND or NOR gates, or three-state elements are arranged in a feedback configuration. NanoTime performs latch timing checks on these structures to ensure that the setup and hold constraints are satisfied.

The **mark_latch** command lets you specify the location of a latch that NanoTime does not recognize automatically. At a minimum, you must specify the latch node. If the **topo_auto_find_latch_clock** variable is set to **false** (the default), you must also specify a clock pin.

The **-inputs** option specifies the input pins or nets to the latch. Alternatively, you can use both the **-rise_inputs** and **-fall_inputs** options to separately specify the input pins or nets that cause the latch output to rise or fall, respectively. This information reduces the effort that NanoTime expends in analyzing the latch and can reduce runtime.

For example, in the case of a simple inverter, you can use the **-rise_inputs** option with an argument that includes the gate of the PMOS pull-up transistor and the **-fall_inputs** option with an argument that includes the gate of the NMOS pull-down transistor. This is equivalent to using the **-inputs** option with an argument list that includes both the PMOS and NMOS transistor gates. This might improve the total runtime. However, for complex circuits, such as a case where the latch input connects to the latch node through a clocked pass gate, you should use the **-inputs** option to allow NanoTime to analyze the circuit.

The **-setup_to** option specifies where NanoTime performs setup checking in the latch circuit. Allowable values are **latch_net** (the default), **input**, or **output**. Specifying **output** causes NanoTime to check for the arrival of data at the latch output, resulting in a more conservative check than using the latch net. Specifying **input** results in a less restrictive check. In the absence of this option, the setup checking point is determined by the **topo_latch_setup_to** variable, which is set to **latch_net** by default.

Similarly, the **-hold_to** option specifies where NanoTime performs hold checking in the latch circuit. Allowable values are **latch_net** (the default) or **input**. Setting it to **input** causes NanoTime to check for the arrival of data at the latch input, resulting in a more conservative check than using the latch net. In the absence of this option, the hold checking point is determined by the **topo_latch_hold_to** variable, which is set to **latch_net** by default.

If you specify feedback elements, NanoTime stops path tracing at each feedback element to prevent looping through the circuit. If you specify feedforward elements, NanoTime simulates the feedforward elements together to get more accurate delay results.

Feedback devices that are also on a valid timing path are called tapped feedback devices. These devices must be identified with the **-tapped_feedback** option to allow path tracing to continue through the non-feedback paths while preserving the feedback property.

By default, NanoTime uses the latch net as the path endpoint for setup and hold checks. You can change the places where NanoTime performs timing checks by using the **set_timing_check_attachment** command.

To cancel the effects of the **mark_latch** command, or to override recognition of an automatically detected latch, use the **erase_latch** command.

COMMAND PHASING

The **mark_latch** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you use the **mark_latch** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

It is preferable to execute a **match_topology** command before using the **mark_latch** command. This practice improves the accuracy of manual topology marking by allowing NanoTime to propagate the clock network and automatically recognize topologies first.

If you modify the clock network after the first **match_topology** command, you might need to force another clock propagation operation before trying to recognize certain latches. To do this, set the **topo_match_topology_reset_clock_and_topology** variable to **true** and execute the **match_topology -force** command to cause NanoTime to repropagate the clock network and rerecognize topologies.

If you use the **mark_latch** command in a lib_topology definition, it should be in the post_match_topology phase point.

EXAMPLES

The following command specifies the location of a latch that NanoTime cannot recognize automatically. The latch node is lat_node, the latch input net is lat_in, the latch output net is lat_out, the feedforward elements are m7 and m8, and the feedback elements are m2, m3, m4, and m6.

```
nt_shell> mark_latch -latch_net lat_node \
          -inputs lat_in -output lat_out \
          -feedforward {m7 m8} \
          -feedback {m2 m3 m4 m6}
```

SEE ALSO

erase_latch(2)
set_timing_check_attachment(2)
topo_latch_hold_to(3)
topo_latch_setup_to(3)
topo_latch_strict_clock_check(3)
topo_latch_find_ts_feedback(3)
topo_latch_find_tapped_feedback(3)
topo_latch_find_input_thru_clocked_fet(3)
topo_latch_enable_logic_propagation(3)
topo_auto_find_latch_feedback_clock(3)
topo_auto_find_latch_clock_thru_port(3)
topo_auto_find_latch_clock(3)
topo_match_topology_reset_clock_and_topology(3)

mark_memory_bitline

Identifies a pair of a memory bitline and its complement in the design.

SYNTAX

```
status mark_memory_bitline
   bitline_pair
   -memory memory_name
```

Data Types

```
   bitline_pair        list
   memory_name         string
```

ARGUMENTS

bitline_pair

A list or collection containing exactly two objects (nets, ports, or pins).

-memory ***memory_name***

Specifies the name of the memory structure that the bitlines are associated with, corresponding to the **-name** argument of the **create_memory** command.

DESCRIPTION

The **mark_memory_bitline** command defines two objects to be a memory bitline and its complement associated with the named memory specified in the **-memory** argument.

The **mark_memory_bitline** command can be executed only in the memory **skip_array** mode analysis. Nanotime ignores this command in a different memory mode of operation.

A single **nmos_bidi** is the only memory bitcell port type supported in the memory **skip_array** mode analysis, so the **mark_memory_bitline** command assumes that the port type of the bitcells connected

to the bitline pair is **nmos_bidi**.

This command may be used inside a foreach_match block to efficiently mark all bitlines in the array.

COMMAND PHASING

The **mark_memory_bitline** command must be executed between the **create_memory** and **check_topology** commands.

EXAMPLES

The following command identifies a memory bitline pair, {bit0 bitb0}, that are associated with the named memory, sa_mem.

```
nt_shell> mark_memory_bitline {bit0 bitb0} -memory sa_mem
```

SEE ALSO

```
create_memory(2)
report_memory(2)
report_bitline(2)
erase_memory_bitline(2)
```

mark_memory_precharge

Identifies a memory precharge structure in the design.

SYNTAX

```
collection mark_memory_precharge
   -precharge_clock elements
   -evaluate_net    net_list
   [-precharge_other cells]
   [-name    name]
```

Data Types

```
   elements       list
   net_list       list
   cells          list
   name           string
```

ARGUMENTS

-precharge_clock *elements*

The elements that are on the memory precharge clock network.

-evaluate_net *net_list*

The evaluate nets of the memory precharge structure. You must specify either one or two nets.

-precharge_other *cells*

Transistors that are part of the precharge topology but are not the devices included in the **-precharge_clock** option.

-name *name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime for memories automatically recognizes a memory precharge structure consisting of two clocked transistors driving a complementary pair of signals. A third device could also be included for an equalizer device between the two signals. The two signals are typically bitlines of a column or sense amp input signals. Any other precharge configuration must be manually marked.

The **mark_memory_precharge** command lets you specify the location of a memory precharge structure. In the command, you must specify the precharge nets of the memory precharge structure and the evaluate nets of the structure.

To cancel the effects of the **mark_memory_precharge** command, or to override recognition of an automatically detected memory precharge structure, use the **erase_memory_precharge** command.

COMMAND PHASING

The **mark_memory_precharge** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_memory_precharge** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a memory precharge structure with precharge clock elements {Mp0 Mp1 Mn2} and evaluate nets bit and bitb.

```
nt_shell> mark_memory_precharge -precharge_clock {Mp0 Mp1 Mp2} \
          -evaluate_net {bit bitb}
```

SEE ALSO

erase_memory_precharge(2)
get_topology(2)

mark_memory_write_circuit

Identifies a memory write circuit structure in the design.

SYNTAX

```
collection mark_memory_write_circuit
   -outputs  net_list
   -transistors elements
   [-enable  pin_list]
   [-inputs  net_list]
   [-name    name]
```

Data Types

```
   net_list      list
   elements      list
   pin_list      list
   name          string
```

ARGUMENTS

-outputs *net_list*

The output nets of the write circuit structure.

-transistors *elements*

The list of transistors to include in the write circuit structure.

-enable *pin_list*

The enable pins of the write circuit structure.

-inputs *net_list*

The input nets of the write circuit structure.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime for memories does not auto recognize memory write circuits. You must define them for correct memory analysis.

The **mark_memory_write_circuit** command lets you specify the location of a memory write circuit structure. In the command, you must specify the data inputs into the memory write circuit, the output nets of the write circuit structure, the enabling clock signals, and the transistors that form one or more channel-connected regions of the memory write topology. The outputs of the memory write circuit might be directly connected to the bitline and bitline complement nets, or alternatively might be connected to the input of a write demultiplexer circuit, which drives the bitline and bitline complement nets.

Command options specify the enable pins of the structure, input nets of the structure, output nets of the structure, and a name for the structure which is an attribute available to the **get_topology** command.

The transistor list should consist of the elements that form the structure needed to drive the write circuit output nets, including enable pin signals and all devices which are driven by the input nets.

To cancel the effects of the **mark_memory_write_circuit** command, use the **erase_memory_write_circuit** command

COMMAND PHASING

The **mark_memory_write_circuit** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_memory_write_circuit** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a memory write circuit with output net bit0, a single transistor Mn0, and enable pin Mn0/g.

```
nt_shell> mark_memory_write_circuit -outputs bit0 -transistors Mn0 \
           -enable Mn0/g
```

SEE ALSO

erase_memory_write_circuit(2)

```
get_topology(2)
```

mark_mux

Identifies a multiplexer structure in the design.

SYNTAX

```
collection mark_mux
   -output_net net
   -select_pins pin_list
   [-constraint_type constraint_type]
   [-name structure_name]
```

Data Types

net	string
pin_list	list
constraint_type	string
structure_name	string

ARGUMENTS

-output_net *net*

The multiplexer output net.

-select_pins *pin_list*

The multiplexer select pins.

-constraint_type *constraint_type*

The type of logic constraint on the select pins; valid values are **one_hot**, **at_most_one_hot**, **one_off**, and **at_most_one_off**.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

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 this information to prevent the tracing of false paths.

The **mark_mux** command lets you specify the location of a multiplexer that NanoTime does not recognize automatically. In the command, you must specify the names of the output net and the selection pins.

You can optionally specify a logic constraint on the multiplexer select pins by using the **-constraint_type** option. Set the constraint to **one_hot**, **at_most_one_hot**, **one_off**, or **at_most_one_off**. NanoTime verifies that the constraint is not violated.

To cancel the effects of the **mark_mux** command, or to override recognition of an automatically detected multiplexer, use the **erase_mux** command.

COMMAND PHASING

The **mark_mux** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_mux** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a multiplexer topology that has an output at net N1 and select pins IN1 and IN2.

```
nt_shell> mark_mux -output_net N1 -select_pins \
          [get_pins -leaf -of_objects [get_nets {IN1 IN2}] \
          -filter "lib_pin_name == G"]
```

SEE ALSO

erase_mux(2)
topo_mux_drive_res_ratio(3)
topo_mux_strict_enable_rules(3)

mark_net

Marks a net in the design with special instructions.

SYNTAX

```
status mark_net
   [-precharge]
   [-check_miller_direction]
   [-ignore_keep_within]
   [-clock_gate_checking force_strict | non_strict]
   [-differential_cross_coupled]
   [-differential_cross_coupled_preset]
   [-differential_cross_coupled_clear]
   [-differential_override_preset pins]
   [-differential_override_clear pins]
   [-transistor_pin_load_model transistor | capacitor]
   [-remove_high_impedance_pessimism_reduction]
   nets
```

Data Types

```
   pins                list
   nets                list
```

ARGUMENTS

-precharge

Marks the specified nets as precharge evaluate nets.

-check_miller_direction

Specifies nets needing miller direction checking when active miller loads are being used.

-ignore_keep_within

The effect of the **-keep_paths_within** option of the **trace_paths** command is suppressed on the specified nets.

-clock_gate_checking force_strict | non_strict

Defines the type of checking that will be carried out for clock gates that drive this net: **force_strict** or

non_strict. This setting determines how clock gate resolution is performed on any clock gate driving this net.

-differential_cross_coupled

Marks the specified differential net as forming a cross-coupled structure such as a level shifter with its differential pair.

-differential_cross_coupled_preset

Marks the specified differential net as forming a cross-coupled structure such as a level shifter with its differential pair and additionally specifies that the differential net clamps to logic 1 when the structure is disabled.

-differential_cross_coupled_clear

Marks the specified differential net as forming a cross-coupled structure such as a level shifter with its differential pair and additionally specifies that the differential net clamps to logic 0 when the structure is disabled.

-differential_override_preset ***pins***

List of transistor gate pins that override the differential behavior of the specified net by setting it to logic 1.

-differential_override_clear ***pins***

List of transistor gate pins that override the differential behavior of the specified net by setting it to logic 0.

-transistor_pin_load_model transistor | capacitor

Controls how the loadings of turned-off transistors which are source or drain connected to the specified net are modeled. The **transistor** option models the load as a full transistor, including all of the nonlinearities in the capacitance. The **capacitor** option models the load as a linear capacitance.

-ignore_high_impedance_pessimism_reduction

Annotates the net to ignore the high_impedance pessimism reduction attribute.

nets

Specifies the nets to be marked.

DESCRIPTION

The **mark_net** command marks one or more nets in the design with special instructions.

The **-precharge** option lets you specify the location of a precharge evaluate net, which guides the automatic topology recognition algorithm to look for precharge structures at that net. The **mark_feedback** command might also be needed to correctly identify all the other devices associated with the precharge net such as precharge clock, evaluate data, and evaluate clock. To specify precharge structures explicitly, use the **mark_precharge** command.

The **-check_miller_direction** option affects the way that NanoTime performs active Miller analysis. When Miller analysis is enabled by setting the **sim_miller_use_active_load** variable to **true**, the tool sets the active load outputs to switch in the opposite direction of the trigger net to get the worst-case effect. However, this assumption might be too pessimistic in cases where the output is switching in the same direction as the trigger net. If the **sim_miller_direction_check** variable is **true**, NanoTime analyzes the switching direction of transmission gates of multiplexer circuits, which can reduce pessimism. To check the switching direction of all other transmission gate circuits, use the **mark_net** command with the **-check_miller_direction** option. If Miller analysis is not enabled, the option has no effect.

The **-ignore_keep_within** option affects the way that path tracing is done on the specified nets. When the **trace_paths** command is issued with the **-keep_paths_within** option, it creates less restrictive pruning. This can lead to long runtime for some circuit configurations. This option can be used to selectively suppress this effect on the specified nets.

The **-clock_gate_checking** option affects the way that NanoTime enforces clock gate checking for clock gates that drive this net. This setting can be used to override the value set by the **topo_clock_gate_strict_checking** variable, which applies to the whole design on a net by net basis. There are two values that can be set with the option. The **force_strict** setting checks for any upstream logic gates that can block the clock arrival to the clock gate at this net. If so, this clock gate is recognized as an unresolved pulse-shaper or unresolved pulse-generator to guarantee pessimism. The **non_strict** setting specifies a more aggressive clock gate recognition and marks the clock gate driving this net to be a resolved pulse-shaper or resolved pulse-generator.

To cancel the effects of the **mark_net** command, use the **erase_net** command.

Command Phasing

The **mark_net** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

The **-clock_gate_checking** option to the **mark_net** command must be called before the **match_topology** command.

EXAMPLES

The following command identifies net Sn12 as a precharge evaluate net.

```
nt_shell> mark_net -precharge Sn12
1
```

SEE ALSO

```
create_lib_topology(2)
erase_net(2)
mark_precharge(2)
```

```
topo_match_topology_reset_clock_and_topology(3)
sim_miller_use_active_load(3)
sim_miller_direction_check(3)
topo_clock_gate_strict_checking(3)
sim_side_transistor_pin_load_model(3)
```

mark_pin

Marks a pin with instructions to ignore the current logic constraints on that pin.

SYNTAX

```
status mark_pin
   [-kill_path_constraints]
   pins
```

Data Types

```
   pins          list
```

ARGUMENTS

-kill_path_constraints

Marks the specified pins with instructions to ignore the current logic constraints.

pins

Specifies the pins in the design to be marked.

DESCRIPTION

The **mark_pin** command marks one or more pins in the design with special instructions. The only available option is the **-kill_path_constraints** option, which specifies to ignore any logic constraints previously set on that pin. Use this option to suppress timing checks on paths through specified pins.

To find the name of a specific pin, use the **report_cell -connections** command.

To cancel the effects of the **mark_pin** command, use the **erase_pin** command.

COMMAND PHASING

The **mark_pin** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command marks pin Xxor7.A with instructions to ignore the current logic constraints.

```
nt_shell> mark_pin -kill_path_constraints Xxor7.A
```

SEE ALSO

erase_pin(2)
report_cell(2)

mark_power_switch

Identifies a power switch structure in the design.

SYNTAX

```
collection mark_power_switch
   -outputs net
   [-inputs net]
   [-transistors element]
   [-name structure_name]
```

Data Types

```
   net                  list
   element              list
   structure_name       string
```

ARGUMENTS

-outputs *net*

The output nets of the power switch.

-inputs *net*

The input net of the power switch.

-transistors *element*

The transistors of the power switch.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes power switch structures based on the user-defined virtual supply, the real supply which drives the virtual supply, and the transistors that are channel-connected to the real supply and the virtual supply.

The real supply is the input to the power switch topology structure and the virtual supply is the output. When the power switch topology is turned off in low power mode, NanoTime does not perform path tracing through the region controlled by the power switch topology.

The **mark_power_switch** command lets you specify the location of a power switch that NanoTime does not recognize automatically. In the command, you must specify the names of the output nets. You can optionally specify the input nets and transistors. NanoTime verifies that the specified outputs are user-defined virtual supply nets and that the inputs are real supply nets.

To cancel the effects of the **mark_power_switch** command, or to override recognition of an automatically detected power switch, use the **erase_power_switch** command.

COMMAND PHASING

The **mark_power_switch** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_power_switch** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a power switch that has an output at virtual supply net vvdd. The input is vdd.

```
nt_shell> mark_power_switch -outputs vvdd -inputs vdd \
          -transistors "mp0 mp1 mp2"
```

SEE ALSO

erase_power_switch(2)
set_supply_net(2)

mark_precharge

Identifies a domino precharge structure and specifies the functions of transistors used in the structure.

SYNTAX

```
collection mark_precharge
   -type  NMOS | PMOS
   [-latch_type latch_type]
   -evaluate_net objects
   -evaluate_data objects
   [-evaluate_clock objects]
   [-evaluate_other objects]
   -precharge_clock objects
   [-precharge_data objects]
   [-precharge_other objects]
   [-feedforwards objects]
   [-keeper_data objects]
   [-keeper_other objects]
   [-setup_to value]
   [-hold_to value]
   [-name structure_name]
   [-match_pattern]
```

Data Types

```
   latch_type           string
   objects              list
   value                string
   structure_name       string
```

ARGUMENTS

-type

The type of transistor used in the evaluate stack, either **NMOS** for a precharge circuit or **PMOS** for a predischarge circuit. The default is **NMOS**. The remaining argument descriptions refer to an NMOS or precharge circuit; the information for a PMOS or predischarge circuit appears in parentheses.

-latch_type

The type of latch construction. The default is **none**. This option determines the timing checks performed on the domino gate. Valid values are as follows:

```
none        No keeper
half_latch  Half-latch keeper
full_latch  Full keeper, different phase from data
retain      Full keeper, same phase as data
flop        Full keeper, same phase as data, flip-flop
```

The **flop**, **latch** and **retain** latch types all have the same full-keeper feedback structure. Because of the pulldown transistor on the evaluate net, a data input that is high does not need to remain high throughout the evaluate phase. This relaxes the falling-clock to falling-input hold constraint. The other timing checks done for the **full_latch** and **retain** cases are the same as those done for the **half_latch** case.

-evaluate_net ***objects***

The precharge node: the net that is precharged (predischarged) during one clock state, then evaluated during the opposite clock state when the evaluate stack conditionally discharges (charges) the net.

-evaluate_data ***objects***

The data-controlled transistors in the evaluate stack. These transistors determine the logical function of the domino structure. The gate of each transistor must be a data signal, not a clock. For an NMOS type evaluate stack, each of these transistors must be NMOS and must be in some directed path from the evaluate net to ground, possibly through an evaluate clock transistor. To determine the evaluate time, NanoTime traces the path from the gate of the transistor to the evaluate net.

-evaluate_clock ***objects***

A clock-controlled transistor that prevents conduction through the evaluate stack during the precharge phase. The gate of the transistor must be a clock signal, possibly a gated clock signal. A footless domino structure does not have an evaluate clock transistor. For an NMOS type evaluate stack, the evaluate clock transistor must also be NMOS and must be in a directed path from the evaluate net to ground. (For a PMOS type evaluate stack, the evaluate clock transistor must be a PMOS transistor that is turned off during the predischarge phase and prevents the evaluate stack from conducting.) To determine the evaluate time, NanoTime traces the path from the gate of the transistor to the evaluate net.

-evaluate_other ***objects***

A weak transistor with a constant gate value that serves as a pullup or pulldown on a net in the evaluate stack. NanoTime does not trace the gate-to-drain path of this transistor.

-precharge_clock ***objects***

The primary precharge device: the clock-controlled transistor that precharges the evaluate net. The gate of the transistor must be a clock signal. For an NMOS type evaluate stack, the precharge clock transistor must be PMOS and must be in a directed path from Vdd to the evaluate net. (For a PMOS type evaluate stack, the precharge clock transistor must be NMOS with a path to ground from the evaluate net.) To determine the precharge time, NanoTime traces the path from the gate of the transistor to the domino gate output.

-precharge_data ***objects***

A data-controlled transistor that precharges an intermediate node in the evaluate stack. NanoTime does not trace the gate-to-drain path of this transistor.

-precharge_other ***objects***

A secondary precharge device: a clock-controlled transistor that precharges an intermediate node in the

evaluate stack, or that performs a weak precharge of the evaluate net. NanoTime does not trace the gate-to-drain path of this transistor.

-feedforwards ***objects***

The elements that feed the value from the evaluate net to the feedback elements.

-keeper_data ***objects***

A data-controlled feedback transistor that holds the current state constant on the evaluate net. NanoTime does not trace the gate-to-drain path of this transistor.

-keeper_other ***objects***

A weak transistor with a constant gate value that serves as a pullup or pulldown on the evaluate net. NanoTime does not trace the gate-to-drain path of this transistor.

-setup_to ***value***

The place in the latch structure where setup checking is to be performed. Allowable values are **evaluate_net** (the default), **output**, and **input**. A value of **output** causes NanoTime to check for the arrival of data at the precharge output, resulting in a more conservative check than using the evaluate net. A value of **input** results in a more relaxed check.

-hold_to ***value***

The place in the latch structure where hold checking is to be performed. Allowable values are **evaluate_net** (the default) or **input**. A value of **input** causes NanoTime to check for the arrival of data at the precharge input, resulting in a more conservative check than using the evaluate net.

-name ***structure_name***

The structure name assigned to the topology, which can be used with the **get_topology** command.

-match_pattern

Builds a pattern with the nets and devices specified in the command and marks all precharge structures that match the same pattern.

DESCRIPTION

NanoTime automatically recognizes a variety of domino precharge structures based on the configuration of transistors, clock signals, and data signals. NanoTime performs precharge timing checks on these structures to ensure that the setup and hold time requirements are met.

NanoTime recognizes precharge and predischarge circuits, footed and footless evaluation structures, several types of half-latch and full-latch feedback structures, and single and multiple evaluation circuits per evaluate clock device (including different clocks on different stacks). The tool can recognize transistor stacks with the clock-controlled and data-controlled transistors in any order.

The **mark_precharge** command lets you specify the location of a domino precharge structure that NanoTime does not recognize automatically. You must identify specific functions of transistors so that NanoTime can perform the appropriate timing checks.

By default, NanoTime uses the latch net as the path endpoint for setup and hold checks. You can change the places where NanoTime performs timing checks by using the **set_timing_check_attachment** command.

For more information about precharge domino structures, including circuit diagrams and the corresponding **mark_precharge** command options, see the NanoTime User Guide.

To cancel the effects of the **mark_precharge** command, or to override recognition of an automatically detected domino precharge structure, use the **erase_precharge** command.

COMMAND PHASING

The **mark_precharge** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If you modify the clock network after the first **match_topology** command, you might need to force another clock propagation operation before trying to recognize or mark precharge structures. To do this, set the **topo_match_topology_reset_clock_and_topology** variable to **true** and execute the **match_topology -force** command to repropagate the clock network and rerecognize topologies.

If the **mark_precharge** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a simple precharge gate having one precharge clock transistor, a half-latch keeper, and a footless stack with two evaluate data transistors.

```
nt_shell> mark_precharge -type NMOS \
          -latch_type half_latch \
          -evaluate_net enode \
          -precharge_clock MP1 \
          -keeper_data MP2 \
          -evaluate_data { MN1 MN2 }
```

The following command identifies a predischarge gate having one predischarge clock transistor, a stack with two evaluate data transistors, and an evaluate clock transistor.

```
nt_shell> mark_precharge -type PMOS \
          -latch_type none \
          -evaluate_net enode \
          -precharge_clock MN1 \
          -evaluate_data { MP1 MP2 } \
          -evaluate_clock MP3
```

The following command identifies a set of two precharge gates that share a common evaluate clock transistor.

```
nt_shell> mark_precharge -type NMOS \
          -latch_type half_latch \
```

```
            -evaluate_net {e1 e2} \
            -precharge_clock {MP11 MP21} \
            -keeper_data {MP12 MP22} \
            -evaluate_data {MN11 MN12 MN21 MN22} \
            -evaluate_clock MN13
```

The following command identifies a precharge gate that has a half-latch keeper and a secondary precharge transistor to precharge an intermediate node in the evaluate stack.

```
nt_shell> mark_precharge -type NMOS \
            -latch_type half_latch \
            -evaluate_net enode \
            -precharge_clock MP1 \
            -precharge_other MP3 \
            -keeper_data MP2 \
            -evaluate_data { MN1 MN2 } \
            -evaluate_clock MN3
```

The following command identifies a precharge gate that has a full keeper and two precharge data transistors to precharge an intermediate node in the evaluate stack.

```
nt_shell> mark_precharge -type NMOS \
            -latch_type latch \
            -evaluate_net enode \
            -precharge_clock MP1 \
            -precharge_data MP3 MP4 \
            -keeper_data {MP2 MN3} \
            -evaluate_data { MN1 MN2 }
```

The following command identifies a precharge gate that has a half-latch keeper on the evaluate net, a secondary precharge transistor to precharge an intermediate node in the evaluate stack, and a full keeper on the intermediate node in the stack.

```
nt_shell> mark_precharge -type NMOS \
            -latch_type half_latch \
            -evaluate_net enode \
            -precharge_clock MP1 \
            -precharge_other MP4 \
            -keeper_data {MP2 MP3 MN3} \
            -evaluate_data { MN1 MN2 }
```

SEE ALSO

```
erase_precharge(2)
set_timing_check_attachment(2)
get_topology(2)
topo_match_topology_reset_clock_and_topology(3)
```

mark_pulldown

Identifies a pulldown transistor, which cannot pass a rising-edge transition to its output source or drain.

SYNTAX

```
collection mark_pulldown
   -transistors elements
   [-name structure_name]
```

Data Types

elements	list
structure_name	string

ARGUMENTS

-transistors *elements*

The list of pulldown transistors to be marked.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

The **mark_pulldown** command lets you specify the location of one or more pulldown transistors in the design, where a rising-edge transition cannot be propagated due to the lack of a pullup transistor.

To cancel the effects of the **mark_pulldown** command, use the **erase_pulldown** command.

COMMAND PHASING

The **mark_pulldown** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the

command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_pulldown** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies transistors mn3 and mn8 as pulldown transistors.

```
nt_shell> mark_pulldown -transistors {mn3 mn8}
```

SEE ALSO

erase_pulldown(2)
mark_pullup(2)
erase_pullup(2)

mark_pullup

Identifies a pullup transistor, which cannot pass a falling-edge transition to its output source or drain.

SYNTAX

```
collection mark_pullup
   -transistors elements
   [-name structure_name]
```

Data Types

```
   elements                list
   structure_name          string
```

ARGUMENTS

-transistors *elements*

The list of pullup transistors to be marked.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

The **mark_pullup** command lets you specify the location of one or more pullup transistors in the design, where a falling-edge transition cannot be propagated due to the lack of a pulldown transistor.

To cancel the effects of the **mark_pullup** command, use the **erase_pullup** command.

COMMAND PHASING

The **mark_pullup** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before

the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_pullup** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies transistors mp3 and mp8 as pullup transistors.

```
nt_shell> mark_pullup -transistors {mp3 mp8}
```

SEE ALSO

erase_pullup(2)
mark_pulldown(2)
erase_pulldown(2)

mark_ram

Identifies a RAM (random-access memory) structure in the design.

SYNTAX

```
collection mark_ram
   -node1 net
   -node2 net
   [-name structure_name]
```

Data Types

```
   net                  string
   structure_name       string
```

ARGUMENTS

-node1 ***net***

Identifies one input net of the RAM structure.

-node2 ***net***

Identifies the other input net of the RAM structure.

-name ***structure_name***

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes a RAM (random-access memory) structure where two equal-strength inverters feed back into each other. NanoTime terminates a path search when it reaches a RAM cell.

The **mark_ram** command lets you specify the location of a RAM structure that NanoTime does not recognize automatically. In the command, you specify the names of the two input nets of the RAM

structure. Note that NanoTime only supports RAM structures that consist of two back-to-back inverters.

To cancel the effects of the **mark_ram** command, or to override recognition of an automatically detected ram structure, use the **erase_ram** command.

COMMAND PHASING

The **mark_ram** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_ram** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a RAM structure having input nodes at nets n47 and n48.

```
nt_shell> mark_ram -node1 n47 -node2 n48
```

SEE ALSO

```
erase_ram(2)
topo_ram_drive_res_ratio(3)
topo_ram_search_thru_cell(3)
topo_ram_find_all_pulldowns(3)
```

mark_register_file

Identifies a register file structure in the design.

SYNTAX

```
collection mark_register_file
   -net1 net
   -net2 net
   -clock1 pin_list
   -clock2 pin_list
   [-transistors transistor_list]
   [-name structure_name]
   [-match_pattern]
```

Data Types

net	string
pin_list	list
transistor_list	list
structure_name	string

ARGUMENTS

-net1 *net*

One data node of the register file structure.

-net2 *net*

The second data node of the register file structure.

-clock1 *pin_list*

The clock pins controlling net1.

-clock2 *pin_list*

The clock pins controlling net2.

-transistors *structure_name*

Additional transistors that are part of the register file.

-name *structure_name*

A structure name assigned to the topology, which can be used with the **get_topology** command.

-match_pattern

Builds a register file pattern with the nets and devices specified in the command and marks all register files that match the same pattern.

DESCRIPTION

NanoTime automatically recognizes a register file structure where two equal-strength inverters feed back into each other and are controlled by two clocks. The output of a register is taken directly from the internal inverters, unlike the output of a RAM, which is taken through a write-signal pass gate. NanoTime does not search through a register cell if the **topo_ram_search_thru_cell** variable is **false** (the default is **true**).

The **mark_register_file** command lets you specify the location of a register file structure that NanoTime does not recognize automatically. The command must specify the names of the nets that are the register data nodes and clock pins.

To cancel the effects of the **mark_register_file** command, or to override recognition of an automatically detected register file structure, use the **erase_register_file** command.

COMMAND PHASING

The **mark_register_file** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_register_file** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a register file structure having data nodes at nets n47 and n48 and controlling clocks at pins m51.g and m52.g.

```
nt_shell> mark_register_file -net1 n47 -net2 n48 \
          -clock1 m51.g -clock2 m52.g
```

SEE ALSO

erase_register_file(2)
topo_regfile_search_all_clock_pins(3)
topo_ram_drive_res_ratio(3)
topo_ram_search_thru_cell(3)
topo_ram_find_all_pulldowns(3)

mark_sense_amp

Identifies a sense amplifier structure in a memory design.

SYNTAX

```
collection mark_sense_amp
   -transistors elements
   -enable      pin
   -inputs      net_list
   [-outputs      net_list]
   [-high_inputs   net_list]
   [-low_inputs    net_list]
   [-diff_high    diff_high_voltage]
   [-diff_low    diff_low_voltage]
   [-name name]
```

Data Types

```
   elements          list
   pin               string
   net_list          list
   diff_high_voltage float
   diff_low_voltage  float
   name              string
```

ARGUMENTS

-transistors *elements*

The user-defined transistors of the sense amplifier.

-enable *pin*

The enable pin of the sense amplifier.

-inputs *net_list*

The input nets of the sense amplifier.

-outputs *net_list*

The output nets of the sense amplifier.

-high_inputs ***net_list***

A list of nets tied to a high logic level during delay calculation.

-low_inputs ***net_list***

A list of nets tied to a low logic level during delay calculation.

-diff_high ***diff_high_voltage***

The differential high voltage to be used as an initial condition on the sense-amp cross-coupled node.

-diff_low ***diff_low_voltage***

The differential low voltage to be used as an initial condition on the sense-amp cross-coupled node.

-name ***structure_name***

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime for memories recognizes a memory sense amplifier structure consisting of two PMOS pullup transistors and two NMOS pulldown transistors cross-coupled to the bitlines.

The **mark_sense_amp** command lets you specify the location of a sense amplifier structure. In the command, you must specify the input nets of the sense amplifier, the enable pin of the structure, and the transistors that form the differential voltage amplifier. Command options specify the output nets of the structure and a name for the structure, which is an attribute available to the **get_topology** command.

The transistor list should consist of the elements of the differential amplifier that connects the input and enable pins to the output nets.

In the memory skip_array mode analysis, both arguments to the -diff_high and -diff_low options must be specified. They will be used as initial conditions on the sense-amp cross-coupled nodes when computing the sense-enable to output path. In other memory modes of operations, Nanotime ignores those arguments.

Timing checks

NanoTime for memories automatically generates differential voltage measurements for sense amplifier topologies. The tool measures the voltage differential between the sense amp inputs when the enable pin reaches 50 percent of VDD. The check value is 10 percent of VDD. Use the **report_measurement** command to view the measurement results.

Use the **erase_sense_amp** command to cancel the effects of the **mark_sense_amp** command.

COMMAND PHASING

The **mark_sense_amp** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command,

NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_sense_amp** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a memory sense-amp structure with transistor list {Mp3 Mp4 Mn0 Mn1 Mnsaen}, input nets bit and bitb, and enable pin Mnsaen/g.

```
nt_shell> [mark_sense_amp -transistors {Mp3 Mp4 Mn0 Mn1 Mnsaen} \
           -inputs {bit bitb} -enable {Mnsaen/g}
```

SEE ALSO

```
erase_sense_amp(2)
get_topology(2)
report_measurement(2)
```

mark_simulation

Specifies a region of the design to be simulated using dynamic delay simulation.

SYNTAX

```
status mark_simulation
   [-channel_expansion_only]
   [-max_inputs num]
   -net nets
```

Data Types

```
   num         integer
   nets        list
```

ARGUMENTS

-channel_expansion_only

Prevents expansion of the dynamic simulation region beyond the channel-connected block containing the specified nets.

-max_inputs *num*

Specifies the maximum number of inputs to the dynamic simulation region. The default is 10 inputs. Specify a larger number to allow a region to have more than 10 inputs, which can result in longer runtime. Specify a lower number to restrict the region definition.

-net *nets*

The net or nets that are to be included in the simulation region.

DESCRIPTION

The **mark_simulation** command specifies the portion of the design to be simulated using dynamic delay

simulation. All transistors with a channel-connected path to the specified net become part of the same simulation region. If multiple nets are specified, adjacent regions are joined into a single region.

If there is an inverter feeding both a signal and its complement into the channel-connected simulation region, NanoTime adds the inverter to the region for better accuracy. An example of this type of circuit is a transmission gate using ctr and ctr_bar signals to control the pass transistors. If you do not want such circuitry added to the simulation region, use the **-channel_expansion_only** option. In that case, only the channel-connected paths containing the specified nets are included in the region.

By default, the maximum number of inputs to a dynamic delay simulation region is 10. If a specified region has more than 10 inputs, NanoTime rejects it and does not simulate it, to prevent excessively long runtime. To specify a larger or smaller limit, use the **-max_inputs** option.

COMMAND PHASING

The **mark_simulation** command must be executed after the **check_topology** command but before the **check_design** command. If you use the **mark_simulation** command after the **check_design** command, NanoTime discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command specifies that nets n18 and n20 are to be simulated with dynamic simulation.

```
nt_shell> mark_simulation -net {n18 n20}
Number of nets specified: 2
mark_simulation: marking net n18 for DSX simulation.
mark_simulation: marking net n20 for DSX simulation.
1
```

NanoTime marks the specified nets for simulation. Each net belongs to a channel-connected region. If the two regions are adjacent, they are joined into a single region for simulation. Otherwise, the two regions are simulated separately.

SEE ALSO

```
all_simulations(2)
get_simulations(2)
report_simulation(2)
set_simulation_attributes(2)
```

mark_tgate

Identifies a transmission gate in the design.

SYNTAX

```
collection mark_tgate
   -n_transistor element
   -p_transistor element
   [-name structure_name]
```

Data Types

```
   element              list
   structure_name       string
```

ARGUMENTS

-n_transistor *element*

The NMOS transistor of the transmission gate structure.

-p_transistor *element*

The PMOS transistor of the transmission gate structure.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes a transmission gate (also known as a tgate or transfer gate) consisting of an NMOS and a PMOS transistor with both their sources and drains connected. NanoTime uses information about transmission gates to correctly generate critical paths and to improve delay calculation accuracy.

By default, NanoTime recognizes NMOS-PMOS transistor pairs as transmission gate even when there is no inverter between the gates of the transistors. To cause recognition to occur only when an inverter is present, set the **topo_tgate_mark_all_pairs** variable to **true**.

The **mark_tgate** command lets you specify the location of a transmission gate that NanoTime does not recognize automatically. In the command, you specify the NMOS transistor and the PMOS transistor in the design that operate as pass transistors.

To cancel the effects of the **mark_tgate** command, or to override recognition of an automatically detected transmission gate, use the **erase_tgate** command.

COMMAND PHASING

The **mark_tgate** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_tgate** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a transmission gate that uses NMOS transistor mn21 and PMOS transistor mp18 as the pass transistors.

```
nt_shell> mark_tgate -n_transistor mn21 \
          -p_transistor mp18
```

SEE ALSO

erase_tgate(2)
topo_tgate_mark_all_pairs(3)

mark_turnoff

Identifies a three-state turnoff structure in the design.

SYNTAX

```
collection mark_turnoff
   -enable_pins pin_list
   -output_net net
   [-name structure_name]
```

Data Types

```
   pin_list              list
   net                   string
   structure_name        string
```

ARGUMENTS

-enable_pins *pin_list*

The enable pins of the turnoff structure.

-output_net *net*

The output net of the turnoff structure.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes a three-state turnoff structure consisting of two PMOS pullup transistors and two NMOS pulldown transistors connected in series, with complementary enable signals at the respective PMOS and NMOS transistor gates.

The **mark_turnoff** command lets you specify the location of a turnoff structure that NanoTime does not recognize automatically. In the command, you must specify the output net of the turnoff structure and the enable pins of the structure, where NanoTime performs timing checks.

To cancel the effects of the **mark_turnoff** command, or to override recognition of an automatically detected turnoff structure, use the **erase_turnoff** command.

COMMAND PHASING

The **mark_turnoff** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_turnoff** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies a three-state turnoff structure with input pins Mn6.g and Mn7.g and output net out1.

```
nt_shell> mark_turnoff -enable_pins {Mn6.g Mn7.g} \
          -output_net out1
```

SEE ALSO

erase_turnoff(2)

mark_weak_pullup

Identifies a weak pullup transistor in the design.

SYNTAX

```
collection mark_weak_pullup
   -transistors element]
   [-name structure_name]
```

Data Types

```
   element              list
   structure_name        string
```

ARGUMENTS

-transistors *element*

The transistor or transistors to be marked as pullup transistors.

-name *pullup_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

The **mark_weak_pullup** command identifies a weak pullup transistor in the design. By default, NanoTime assumes that any transistor that is turned on and connected to Vdd is a weak pullup that pulls the net high if there is nothing pulling the net down.

The **mark_weak_pullup** command lets you specify the location of a pullup structure that NanoTime does not recognize automatically. In the command, you specify the transistors in the design that are to be marked as pullup transistors.

To cancel the effects of the **mark_weak_pullup** command, or to override recognition of an automatically

detected weak pullup transistor, use the **erase_weak_pullup** command.

COMMAND PHASING

The **mark_weak_pullup** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_weak_pullup** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies transistor m1 as a weak pullup transistor.

```
nt_shell> mark_weak_pullup m1
```

SEE ALSO

```
erase_weak_pullup(2)
topo_weak_pullup_n_length(3)
topo_weak_pullup_n_width(3)
topo_weak_pullup_p_length(3)
topo_weak_pullup_p_width(3)
```

mark_xor

Identifies an exclusive OR structure in the design.

SYNTAX

```
collection mark_xor
   -output_net net
   -input_pins pin_list
   [-name structure_name]
```

Data Types

```
   net              string
   pin_list         list
   structure_name   string
```

ARGUMENTS

-output_net *net*

The output net of the XOR structure.

-input_pins *pin_list*

The input pins of the XOR structure.

-name *structure_name*

The structure name assigned to the topology, which can be used with the **get_topology** command.

DESCRIPTION

NanoTime automatically recognizes a particular 6-transistor implementation of XOR (exclusive OR) and XNOR (exclusive NOR) logic gates. NanoTime uses this information to eliminate false paths for certain logic states.

The **mark_xor** command lets you specify the location of an XOR structure that NanoTime does not recognize automatically. In the command, you specify the output net and input pins of the XOR structure.

To cancel the effects of the **mark_xor** command, or to override recognition of an automatically detected XOR or XNOR structure, use the **erase_xor** command.

COMMAND PHASING

The **mark_xor** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). You cannot use the command before the **link_design** command. If you execute it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If the **mark_xor** command is used in a lib_topology definition, it should be in the pre_match_topology or post_match_topology phase point.

EXAMPLES

The following command identifies an XOR structure with an output net out1 and input pins mn6 and mn7.

```
nt_shell> mark_xor -output_net out1 -input_pins {mn6.g mn7.g}
```

SEE ALSO

erase_xor(2)

match_topology

Runs the NanoTime topology recognition algorithm.

SYNTAX

```
status match_topology
   [-name structure_name]
   [-force]
   [-structure_types type_list]
   [-message_level level]
```

Data Types

```
   structure_name    string
   type_list         list
   level             string
```

ARGUMENTS

-name *structure_name*

Assigns an optional name to the structures found and marked by the command. The name can be used with the **get_topology** command.

-force

Forces NanoTime to repropagate the clock network and rerecognize topologies if the **topo_match_topology_reset_clock_and_topology** variable is **true**.

-structure_types *type_list*

Specifies the types of structures to look for. If this option is not used, NanoTime looks for the structure types listed in the **topo_auto_search_class** variable.

-message_level *level*

Specifies the amount of information reported in the terminal window. The allowed values are **silent**, **normal** (the default), **verbose**, and **debug**.

DESCRIPTION

The **match_topology** command runs the NanoTime topology recognition algorithm. NanoTime examines the configuration of transistors in the design to identify and mark various circuit structures such as latches, inverters, and multiplexers. NanoTime needs this information so that it can perform timing checks at the appropriate structures in the design.

The **-structure_types** option of the **match_topology** command specifies a list of the types of structures that NanoTime looks for. These are the structures that can be listed in the option:

```
clock_gate            flip_flop          register
cross_coupled         mux                tgate
cross_coupled_pmos    precharge          turnoff
feedback              pulldown           weak_pullup
inverter              pullup             xor
latch                 ram                differential_synchronizer
```

NanoTime looks for the listed structures, plus any additional structures necessary to identify them. For example, if you specify the **-structure_types precharge** option, NanoTime also looks for and marks clock gate, feedback, and latch structures because the tool needs to identify those other structures before it can identify precharge structures.

If the **-structure_types** option is not used in the **match_topology** command, NanoTime looks for the structure types listed in the **topo_auto_search_class** variable. By default, that variable is set to the following string:

```
mux clock_gate turnoff xor cross_coupled nand nor latch precharge ram feedback
weak_pullup differential_synchronizer
```

After the **match_topology** command, you can use the **report_topology** command to report the number of structures of each type found and marked by the **match_topology** command, as well as the locations of those structures. You can use the **erase_*** and **mark_*** commands, possibly in conjunction with the **foreach_match** command, to override the marking of structures found or not found by the **match_topology** command.

You might need to execute the **match_topology** command more than one time in a complex flow. For example, after recognizing or marking clock gating structures, a clock propagation step might be needed to allow dependent structures such as latches to be recognized properly. To force NanoTime to repropagate clocks and rerecognize topologies, set the **topo_match_topology_reset_clock_and_topology** variable to **true**, then execute the **match_topology** command with the **-force** option. For more information, see the *NanoTime User Guide*.

After all structures have been properly identified and marked in the design, perform a final check with the **check_topology** command.

COMMAND PHASING

The **match_topology** command can only be executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command). If you use the **match_topology** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following command causes NanoTime to look for and mark feedback structures in the design.

```
nt_shell> match_topology -structure_types feedback
```

The following command causes NanoTime to look for and mark all structure types listed in the **topo_auto_search_class** variable.

```
nt_shell> match_topology
```

SEE ALSO

```
create_clock(2)
check_topology(2)
foreach_match(2)
report_topology(2)
topo_auto_search_class(3)
topo_match_topology_reset_clock_and_topology(3)
```

mem

Retrieves the total memory allocated by the current nt_shell process.

SYNTAX

```
status mem
```

DESCRIPTION

The **mem** command returns the size of memory currently allocated by the current nt_shell process for the purposes of storing design netlists, design annotations, and constraints information as well as computed timing information. The reported value represents the amount in kilobytes (kB).

Note that the reported value would not match values obtained by the user through Unix process tracking commands, such as top. This is the case because the **mem** command does not track memory used to load the executable and maintain process stack space. Also, it does not differentiate between resident and swapped memory.

EXAMPLES

```
nt_shell> mem
8092
```

SEE ALSO

cputime(2)

merge_models

Merges multiple timing models in .db format into a single model.

SYNTAX

```
status merge_models
   -db_files model_files
   -mode_names mode_names
   -output output_file_name
   [-group_name mode_group_name]
   [-tolerance merge_tolerance]
   [-percent_tolerance merge_percent_tolerance]
   [-single_mode]
```

Data Types

```
   model_files              list
   mode_names               list
   output_file_name         string
   mode_group_name          string
   merge_tolerance          float
   merge_percent_tolerance  float
```

ARGUMENTS

-db_files *model_files*

Specifies the list of .db model files to merge.

-mode_names *mode_names*

Specifies the list of mode names for the models. The mode names must match the list of model files. Note that models that already have mode statements cannot be merged with the **merge_models** command.

-output *output_file_name*

Specifies the name of the output .db file used to save the merged model.

-group_name *mode_group_name*

Specifies the name of the mode group to which the models belong. If you do not specify this option,

the default group name is **etm_modes**.

-tolerance ***merge_tolerance***

Specifies the tolerance used to compare delay and transition times of arcs in the timing models. It must be a floating-point number greater than or equal to zero. When the timing arcs in the models are matched and compared, they are considered equal if the difference between them is within this tolerance. The default tolerance is 0.04 time units.

-percent_tolerance ***merge_percent_tolerance***

Specifies the percent tolerance used to compare delay and transition times of arcs in the timing models. It must be a floating-point number greater than or equal to zero. When two timing arcs in the models are matched and compared, they are considered equal if the difference between them is within this percent tolerance of both arc numbers, or the absolute tolerance defined by **-tolerance** is satisfied.

-single_mode

Forces each timing arc in the merged model to have only one defined mode, typically resulting in a larger, less compact timing model. This might be necessary for downstream tools that cannot handle more than one mode on a timing arc.

DESCRIPTION

This command merges multiple .db timing models into a single model. The generated model has moded timing arcs such that, in any given mode, the static timing behavior of the model is equivalent to one of the original models that was merged. The final merged model is saved in the specified file. The Synopsys PrimeTime and Design Compiler tools recognize the merged model with moded timing arcs for performing timing analysis.

Block modes

A block can have different operating modes, such as test mode and normal operating mode. The timing requirements for a block can be quite different for different operating modes. In these cases, you need to extract a separate model for each mode.

Instead of keeping and using multiple extracted models, you can optionally merge them into a single model that has different timing arcs enabled for different operating modes. Then you can use this single model and change its behavior by setting it into different modes.

Using the merge_models command

In the **merge_models** command, you specify the .db files of the models to be merged, the names of the modes corresponding to those models, the name of the new model being generated, an optional mode group name, and an optional tolerance value.

The tolerance value specified in the **-tolerance** option specifies how far apart two timing values can be to merge them into a single timing arc. If the corresponding arc delays in two timing models are within this tolerance value, the two arcs are considered the same and are merged into a single arc that applies

to both operating modes. The default tolerance value is 0.04 time units.

When you merge models with a wide range of delay values, an absolute tolerance value might not be appropriate. Use the **-percent_tolerance** option to define the maximum allowable percentage difference between two timing arcs. If both options are specified, the larger of the two tolerance values is used to determine if two timing arcs are considered equal. Note that the absolute tolerance value of 0.04 time units is always present even if the **-tolerance** option is not specified.

The PrimeTime tool can handle more than one mode per timing arc, but some tools cannot. To generate a merged model that can work with all tools, use the **-single_mode** option with the **merge_models** command. This forces the merged model to have no more than one mode per timing arc, resulting in a larger, less compact timing model.

When merging models with multiple power or ground supplies, pins that are not associated with a power or ground pin are connected to power or ground following the default selection rules of the Library Compiler tool. You can avoid using the default connections by explicitly connecting all pins in your design.

COMMAND PHASING

The **merge_models** command is not phase-restricted.

SEE ALSO

`extract_model(2)`

parallel_execute

Specifies a list of commands to be executed in parallel and their output files.

SYNTAX

```
status parallel_execute
   [-compress]
   [-wait]
   concurrent_cmds
```

Data Types

```
   concurrent_cmds     list
```

ARGUMENTS

-compress

If redirecting to a file, this option specifies that the output should be compressed as it is written. The output file is in a format recognizable by "gzip -d".

-wait

Wait for the spawned processes to terminate before continuing.

concurrent_cmds

The list of interleaved strings that consist of commands and their output files. These commands are to be executed in parallel with their results written into the specified output files.

DESCRIPTION

This command enables you to exploit parallelism across unrelated NanoTime Tcl commands. Background processes do not require additional NanoTime licenses and do not require setting up distributed processing protocols.

The command instructs NanoTime to execute a list of NanoTime commands in parallel. The output of each command is written to its specified output file. If you specify options for a command, you must format the command and its options as a string enclosed in double quotation marks. The output file is not inside the quotation marks.

Do not use the > or >> redirection operators inside the **parallel_execute** command.

The **parallel_execute** command uses the following syntax:

```
trace_paths
parallel_execute {
  report_cmd1  rpt_file1
  report_cmd2  rpt_file2
  ...          ...
  report_cmdN  rpt_fileN
}
```

Any report_*, write_*, save_* or restore_* command or custom Tcl reporting procedure is applicable.

You can include up to 1024 commands inside the **parallel_execute** command. NanoTime automatically schedules these commands for immediate execution. If any of the parallel processes fail to spawn successfully, an informational message appears in the parent NanoTime process. The specified command is then executed in the parent process.

For each spawned process, an additional file named by appending ".proc" to the output file name contains information about the spawned process.

The **parallel_execute** command is a non-blocking command. To achieve close to optimal speedup, specify the longest runtime commands first in the list.

EXAMPLES

The following example creates two path reports in parallel.

```
parallel_execute {
   "report_paths -min" min.rpt
   "report_paths -max" max.rpt   }
```

SEE ALSO

```
redirect(2)
pexe_wait_for_exit(3)
```

parse_proc_arguments

Parses the arguments passed into a Tcl procedure.

SYNTAX

```
string parse_proc_arguments -args arg_list
result_array

list arg_list
string result_array
```

ARGUMENTS

-args *arg_list*

Specified the list of arguments passed in to the Tcl procedure.

result_array

Specifies the name of the array into which the parsed arguments should be stored.

DESCRIPTION

The **parse_proc_arguments** command is used within a Tcl procedure to enable use of the -help option, and to support argument validation. It should typically be the first command called within a procedure. Procedures that use **parse_proc_arguments** will validate the semantics of the procedure arguments and generate the same syntax and semantic error messages as any application command (see the examples that follow).

When a procedure that uses **parse_proc_arguments** is invoked with the -help option, **parse_proc_arguments** will print help information (in the same style as using **help -verbose**) and will then cause the calling procedure to return. Similarly, if there was any type of error with the arguments (missing required arguments, invalid value, and so on), **parse_proc_arguments** will return a Tcl error and the calling procedure will terminate and return.

If you didn't specify -help, and the specified arguments were valid, the array variable *result_array* will contain each of the argument values, subscripted with the argument name. Note that the argument name here is NOT the names of the arguments in the procedure definition, but rather the names of the arguments as defined using the **define_proc_attributes** command.

The **parse_proc_arguments** command cannot be used outside of a procedure.

EXAMPLES

The following procedure shows how **parse_proc_arguments** is typically used. The argHandler procedure parses the arguments it receives. If the parse is successful, argHandler prints the options or values actually received.

```
proc argHandler args {
  parse_proc_arguments -args $args results
  foreach argname [array names results] {
    echo "  $argname = $results($argname)"
  }
}
define_proc_attributes argHandler -info "argument processor" \
  -define_args \
  {{-Oos "oos help" AnOos one_of_string {required value_help {values {a b}}}}
   {-Int "int help" AnInt int optional}
   {-Float "float help" AFloat float optional}
   {-Bool "bool help" "" boolean optional}
   {-String "string help" AString string optional}
   {-List "list help" AList list optional}}
  {-IDup int dup AIDup int {optional merge_duplicates}}}
```

Invoking argHandler with the -help option generates the following:

```
prompt> argHandler -help
Usage: argHandler    # argument processor
        -Oos AnOos              (oos help:
                                 Values: a, b)
        [-Int AnInt]            (int help)
        [-Float AFloat]         (float help)
        [-Bool]                 (bool help)
        [-String AString]       (string help)
        [-List AList]           (list help)
```

Invoking argHandler with an invalid option causes the following output (and a Tcl error):

```
prompt> argHandler -Int z
Error: value 'z' for option '-Int' not of type 'integer' (CMD-009)
Error: Required argument '-Oos' was not found (CMD-007)
```

Invoking argHandler with valid arguments generates the following:

```
prompt> argHandler -Int 6 -Oos a
  -Oos = a
  -Int = 6
```

SEE ALSO

define_proc_attributes(2)
help(2)
proc(2)

print_message_info

Prints information about diagnostic messages that have occurred or have been limited.

SYNTAX

```
string print_message_info
   [-ids id_list]
   [-summary]
```

Data Types

```
   id_list      list
```

ARGUMENTS

-ids *id_list*

Specifies a list of message identifiers to report. Each entry can be a specific message or a glob-style pattern that matches one or more messages. If this option is omitted and no other options are given, then all messages that have occurred or have been limited are reported.

-summary

Generates a summary of error, warning, and informational messages that have occurred so far.

DESCRIPTION

The **print_message_info** command enables you to print summary information about error, warning, and informational messages that have occurred or have been limited with the **set_message_info** command. For example, if the following message is generated, information about it is recorded:

Error: unknown command 'wrong_command' (CMD-005)

It is useful to be able to summarize all recorded information about generated diagnostic messages. Much

of this can be done using the **get_message_info** command, but you need to know a specific message ID. By default, **print_message_info** summarizes all of the information. It provides a single line for each message that has occurred or has been limited, and one summary line that shows the total number of errors, warnings, and informational messages that have occurred so far. If an *id_list* is given, then only messages matching those patterns are displayed. If **-summary** is given, then a summary is displayed.

Using a pattern in the *id_list* is intended to show a specific message prefix, for example, "CMD*". Note that this does not show all messages with that prefix. It shows only the messages that have occurred or have been limited.

EXAMPLES

The following example uses **print_message_info** to show a few specific messages:

```
prompt> print_message_info -ids [list "CMD*" APP-99]

Id               Limit    Occurrences   Suppressed
-------------------------------------------------------
CMD-005              0              7            2
APP-99               1              0            0
```

At the end of the session, you might want to generate some information about a set of interesting messages, such as how many times each occurred (which includes suppressions), how many times each was suppressed, and whether a limit was set for any of them. The following example uses **print_message_info** to get this information:

```
prompt> print_message_info

Id               Limit    Occurrences   Suppressed
-------------------------------------------------------
CMD-005              0             12            0
APP-027            100            150           50
APP-99               0              1            0

Diagnostics summary: 12 errors, 150 warnings, 1 informational
```

Note that the suppressed count is not necessarily the difference between the limit and the occurrences, since the limit can be dynamically changed with the **set_message_info** command.

SEE ALSO

get_message_info(2)
set_message_info(2)
suppress_message(2)

print_suppressed_messages

Displays an alphabetical list of message IDs that are currently suppressed.

SYNTAX

```
string print_suppressed_messages
```

ARGUMENTS

The **print_suppressed_messages** command has no arguments.

DESCRIPTION

The **print_suppressed_messages** command displays all messages that you suppressed using the **suppress_message** command. The messages are listed in alphabetical order. You only can suppress informational and warning messages. The result of **print_suppressed_messages** is always the empty string.

EXAMPLES

The following example shows the output from the **print_suppressed_messages** command:

```
prompt> print_suppressed_messages
No messages are suppressed

prompt> suppress_message {XYZ-001 CMD-029 UI-1}

prompt> print_suppressed_messages
The following 3 messages are suppressed:
```

```
CMD-029, UI-1, XYZ-001
```

SEE ALSO

```
suppress_message(2)
unsuppress_message(2)
```

printenv

Prints the value of environment variables.

SYNTAX

```
string printenv
   [variable_name]
```

Data Types

```
   variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of a single environment variable to print.

DESCRIPTION

The **printenv** command prints the values of the environment variables inherited from the parent process or set from within the application with the **setenv** command. When no arguments are specified, all environment variables are listed. When a single *variable_name* is specified, if that variable exists, its value is printed. There is no useful return value from **printenv**.

To retrieve the value of a single environment variable, use the **getenv** command.

EXAMPLES

The following examples show the output from the **printenv** command:

```
prompt> printenv SHELL
/bin/csh

prompt> printenv EDITOR
emacs
```

SEE ALSO

getenv(2)
setenv(2)

printvar

Prints the values of one or more variables.

SYNTAX

```
string printvar
   [pattern]
   [-user_defined | -application]
```

Data Types

```
   pattern      string
```

ARGUMENTS

pattern

Prints variable names that match *pattern*. The optional *pattern* argument can include the wildcard characters * (asterisk) and ? (question mark). If not specified, all variables are printed.

-user_defined

Shows only user-defined variables. This option is mutually exclusive with the **-application** option.

-application

Shows only application variables. This option is mutually exclusive with the **-user_defined** option.

DESCRIPTION

The **printvar** command prints the values of one or more variables. If the *pattern* argument is not specified, the command prints out the values of application and user-defined variables. The **-user_defined** option limits the variables to those that you have defined. The **-application** option limits the variables to those created by the application. The two options are mutually exclusive and cannot be used together.

Some variables are suppressed from the output of **printvar**. For example, the Tcl environment variable array *env* is not shown. To see the environment variables, use the **printenv** command. Also, the Tcl variable **errorInfo** is not shown. To see the error stack, use the **error_info** command.

EXAMPLES

The following command prints the values of all of the variables:

```
prompt> printvar
```

The following command prints the values of all application variables that match the pattern *sh*a**:

```
prompt> printvar -application sh*a*
sh_arch                 = "sparcOS5"
sh_command_abbrev_mode  = "Anywhere"
sh_enable_page_mode     = "false"
sh_new_variable_message = "false"
sh_new_variable_message_in_proc = "false"
sh_source_uses_search_path = "false"
```

The following command prints the value of the **search_path** variable:

```
prompt> printvar search_path
search_path             = ".  /designs/newcpu/v1.6  /lib/cmos"
```

The following command prints the values of the user-defined variables:

```
prompt> printvar -user_defined
a                   = "6"
designDir           = "/u/designs"
```

SEE ALSO

```
error_info(2)
printenv(2)
setenv(2)
```

proc_args

Displays the formal parameters of a procedure.

SYNTAX

```
string proc_args
   proc_name
```

Data Types

```
   proc_name      string
```

ARGUMENTS

proc_name

Specifies the name of the procedure.

DESCRIPTION

The **proc_args** command is used to display the names of the formal parameters of a user-defined procedure. This command is essentially a synonym for the Tcl builtin command **info** with the *args* argument.

EXAMPLES

This example shows the output of **proc_args** for a simple procedure:

```
prompt> proc plus {a b} { return [expr $a + $b] }
```

```
prompt> proc_args plus
a b

prompt> info args plus
a b

prompt>
```

SEE ALSO

info(2)
proc(2)
proc_body(2)

proc_body

Displays the body of a procedure.

SYNTAX

```
string proc_body
   proc_name
```

Data Types

```
   proc_name      string
```

ARGUMENTS

proc_name

Specifies the name of the procedure.

DESCRIPTION

The **proc_body** command is used to display the body (contents) of a user-defined procedure. This command is essentially a synonym for the Tcl builtin command **info** with the *body* argument.

EXAMPLES

This example shows the output of **proc_body** for a simple procedure:

```
prompt> proc plus {a b} { return [expr $a + $b] }

prompt> proc_body plus
```

```
    return [expr $a  + $ b]

prompt> info body plus
    return [expr $a  + $ b]

prompt>
```

SEE ALSO

info(2)
proc(2)
proc_args(2)

query_objects

Searches for and displays objects in the database.

SYNTAX

```
status query_objects
   [-verbose]
   [-class class_name]
   [-truncate elem_count]
   object_spec
```

Data Types

class_name	string
elem_count	integer
object_spec	list

ARGUMENTS

-verbose

Displays the class of each object found. By default, only the name of each object is listed. With this option, each object name is preceded by its class, as in "cell:U1/U3."

-class *class_name*

Establishes the class for a named element in the *object_spec*. Valid classes are design, cell, net, and so on.

-truncate *elem_count*

Truncates display to *elem_count* elements. By default, up to 100 elements are displayed. To see more or fewer elements, use this option. To see all elements, set *elem_count* to 0.

object_spec

Provides a list of objects to find and display. Each element in the list is either a collection or an object name. Object names are explicitly searched for in the database with class *class_name*.

DESCRIPTION

The **query_objects** command finds and displays objects in the runtime database.

The *object_spec* is a list containing collections, object names, or both. For elements of the *object_spec* that are collections, the **query_objects** command displays the contents of the collection.

For elements of the *object_spec* that are names or wildcard patterns, the **query_objects** command searches for the objects in the class specified by *class_name*. Note that the **query_objects** command does not have a predefined order in which to search the classes. If you do not specify *class_name*, only those elements that are collections are displayed. Messages are displayed for the other elements (see EXAMPLES).

To control the number of elements displayed, use the *-truncate* option. If the display is truncated, you see the ellipsis (...) as the last element. If the default truncation occurs, a message shows the total number of elements that would be displayed without truncation (see EXAMPLES).

NOTE: The output of the **query_objects** command looks similar to the output of any command that creates a collection, but remember that the result of the **query_objects** command is always the empty string.

COMMAND PHASING

The **query_objects** command is not phase-restricted.

EXAMPLES

The following examples show the basic usage of the **query_objects** command.

```
nt_shell> query_objects [get_cells o*]
{"or1", "or2", "or3"}
nt_shell> query_objects -class cell U*
{"U1", "U2"}
nt_shell> query_objects -verbose -class cell [list U* [get_nets n1]]
{"cell:U1", "cell:U2", "net:n1"}
```

When you omit the **-class** option, only those elements of the *object_spec* that are collections generate output. The other elements generate error messages.

```
nt_shell> query_objects \[list U* [get_nets n1] n*]
Error: No such collection 'U*' (SEL-001)
Error: No such collection 'n*' (SEL-001)
{"n1"}
```

When the output is truncated, you see the ellipsis at the end of the display. For the following example, assume the default truncation is 5 (it is actually 100).

```
nt_shell> query_objects [get_cells o*] -truncate 2
{"or1", "or2", ...}
nt_shell> query_objects [get_cells *]
{"or1", "or2", "or3", "U1", "U2", ...}
```

```
Output truncated (total objects 126)
```

SEE ALSO

collections(2)
get_cells(2)
get_clocks(2)
get_designs(2)
get_generated_clocks(2)
get_lib_cells(2)
get_lib_pins(2)
get_libs(2)
get_nets(2)
get_pins(2)
get_ports(2)
get_timing_paths(2)
get_topology(2)

quit

Exits the shell.

SYNTAX

```
string quit
```

ARGUMENTS

The **quit** command has no arguments.

DESCRIPTION

The **quit** command exits from the application. It is basically a synonym for the **exit** command with no arguments.

EXAMPLES

The following example exits the current session:

```
prompt> quit
```

SEE ALSO

exit(2)

read_device_parameters

Reads device parameters from a .dpf file and uses the information to annotate the current design.

SYNTAX

```
status read_device_parameters
   [-dpf_scale_factor factor]
   [-path prefix]
   file_names
```

Data Types

```
   factor            float
   prefix            string
   file_names        list
```

ARGUMENTS

-dpf_scale_factor *factor*

Specifies a positive floating-point number to use as a scale factor for the geometrical parameters of DPF transistors. The default is 1.0.

-path *prefix*

Specifies a relative path from the current design to the hierarchical design name for which the device parameter file has been created. By default, absolute path names are used. Use this option if the device parameter file refers to an object in a hierarchy (for example, *hier*). Do not use this option if the device parameter file refers to an absolute path.

file_names

Specifies the names of one or more .dpf device parameter files.

DESCRIPTION

The **read_device_parameters** command reads device parameter data from a file in .dpf format and annotates the information on transistors in the current design, thereby allowing more accurate delay calculation.

To report annotated device parameters that have been read in with the **read_device_parameters** command, use the **report_annotated_device_parameters** command. To remove the annotated parameters, use the **remove_annotated_device_parameters** command. The **reset_design** command removes all attributes from the current design, including annotated device parameters.

COMMAND PHASING

The **read_device_parameters** command can be executed in any state after "netlist-linked" up to and including "topology-checked" (in other words, after the **link_design** command but before the **check_design** command.

EXAMPLES

The following command reads the device parameter file adder.dpf and uses the data to annotate the device parameters on the current design.

```
nt_shell> read_device_parameters adder.dpf
```

The following command removes annotated device parameters from the current design.

```
nt_shell> remove_annotated_parasitics
```

SEE ALSO

```
read_parasitics(2)
remove_annotated_device_parameters(2)
report_annotated_device_parameters(2)
report_cell(2)
reset_design(2)
```

read_lib

Reads in a Synopsys library (.lib) file in Liberty format. The Library Compiler tool must be installed to enable this command.

SYNTAX

```
status read_lib
   file_names
```

Data Types

```
   file_names     list
```

ARGUMENTS

file_names

Specifies the name of a .lib library file to read. The file name must be in the form of *library_name*.lib, where *library_name* is the name of the library defined in the library statement at the beginning of the .lib file.

DESCRIPTION

The **read_lib** command reads a Synopsys library (.lib) file in Liberty format. You must install the Synopsys Library Compiler tool to enable this command. For more information, see the NanoTime installation notes. If you do not install the Library Compiler tool, you must read a library in compiled (.db) format by using the **read_library** command.

To locate files that have relative path names, the NanoTime tool searches for each of the files in the **read_lib** argument list in each directory listed in the **search_path** variable. Files that have absolute path names are loaded directly, regardless of the **search_path** variable setting.

When you use the **read_lib** command, set the **link_path** variable to include the name of the cell contained in the library.

EXAMPLES

In the following example, the file alu_ccs.lib is read based on the **search_path** variable.

```
nt_shell> set search_path "/designs /libs/cmos"

nt_shell> read_lib alu_ccs.lib
1

nt_shell> set link_path " * alu_ccs $link_path"
* alu_ccs $link_path
nt_shell> register_netlist -format verilog alu_ccs.v
1
nt_shell> link_design ALU
 ...
```

SEE ALSO

list_libs(2)
read_library(2)
link_path(3)
search_path(3)

read_library

Reads in Synopsys library database (.db) files containing timing models. This command is equivalent to the **read_db** command.

SYNTAX

```
status read_library
   file_names
```

Data Types

```
   file_names     list
```

ARGUMENTS

file_names

Specifies the names of one or more .db files to read. Typically one file is read.

DESCRIPTION

The **read_library** command reads timing model information from Synopsys database (.db) files, such as those created by the **extract_model** command.

A .db file might contain netlist information, but NanoTime does not use this information. To read in netlists, use the **register_netlist** and **link_design** commands.

Each .db file can be specified as an absolute path (starting with a slash character) or a relative path. When you specify a relative path name, NanoTime searches for each file in each directory listed in the **search_path** variable and uses the first one found. To learn the path for a particular relative path name, use the **which** command. When you specify an absolute path name in the **read_library** command, NanoTime ignores the **search_path** setting. You can use wildcard characters to specify multiple files.

When you use the **read_library** command, set the **link_path** variable to include the name of the cell contained in the library.

Each file can contain one library object. After the files are loaded, use the **list_libs** command to view a list of the library objects.

COMMAND PHASING

The **read_library** command is not phase-restricted.

EXAMPLES

In the following example, the file mycell.db is read in from the /libs/cmos directory, as specified by the **search_path** variable.

```
nt_shell> set search_path "/libs/cmos"
/libs/cmos

nt_shell> read_library mycell.db
Loading db file '/libs/cmos/mycell.db'
1
nt_shell> set link_path " * mycell $link_path"
* mycell $link_path
```

SEE ALSO

```
read_db(2)
list_libs(2)
link_path(3)
search_path(3)
```

read_parasitics

Reads net parasitics information from a DSPF or SPEF file and uses that information to annotate the current design.

SYNTAX

```
status read_parasitics
   [-format SPEF | DSPF ]
   [-complete_with zero | rc]
   [-increment]
   [-path prefix]
   [-quiet]
   [-syntax_only]
   [-keep_capacitive_coupling]
   [-coupling_reduction_factor r_factor]
   [-triplet_type min | typ | max]
   [-dpf_scale_factor s_factor]
   file_names
```

Data Types

```
prefix              string
r_factor            float
s_factor            float
file_names          string
```

ARGUMENTS

-format SPEF | DSPF

Specifies the format of the parasitic data file: SPEF (Standard Parasitic Exchange Format) or DSPF (Detailed Standard Parasitic Format).

-complete_with zero | rc

Causes NanoTime to complete any net at a hierarchical pin that has partially annotated parasitics by inserting parasitic capacitors and resistors. The **zero** value completes partially annotated nets by inserting capacitors and resistors with very small (near-zero) values. The **rc** value completes partially annotated nets with capacitor and resistor values determined by the **parasitics_completion_capacitance** and **parasitics_completion_resistance** variables.

-increment

Causes NanoTime to keep, rather than overwrite, existing parasitic data previously annotated on nets listed in the parasitic data file. Use this option to annotate hierarchical parasitic data files. In the absence of this option, the RC annotations specified in the parasitic data file overwrite the previous parasitics annotated on the nets listed in the file.

-path *prefix*

Specifies a relative path from the current design to the hierarchical design name for which the parasitics file has been created. By default, absolute path names are used. Use this option if the parasitic data file refers to an object (for example, *net*) in a hierarchy (for example, *hier*). Do not use this option if the parasitic data file refers to an absolute path (for example, *hier/net*).

-quiet

Prevents generation of the annotation report. By default, after reading the parasitics file, NanoTime generates a report on the annotated parasitics in the design (including layout-only nets). The **-quiet** option stops this report from being generated. You might want to do this when you are reading multiple parasitic data files incrementally.

-syntax_only

Causes the **read_parasitics** command to check the contents of the parasitic data file for syntax errors without performing parasitic annotation. Use this option to troubleshoot the parasitic data file and avoid error messages that might otherwise occur during actual annotation of a design.

-keep_capacitive_coupling

Causes cross-coupling capacitors to be kept in the RC network data structure, thereby allowing crosstalk analysis to be performed. This option applies to all parasitics file formats (SPEF and DSPF). A NanoTime Ultra license is required. This option cannot be used with the **-coupling_reduction_factor** option and overrides any previous reduction factor setting. If crosstalk analysis is not enabled, all coupling capacitors are split to ground with a factor of 1.0.

-coupling_reduction_factor *r_factor*

Specifies the factor to apply when reducing coupling capacitors to grounded capacitors. It must be a positive floating-point number and the default is 1.0. This option applies to all parasitics file formats (SPEF and DSPF). It cannot be used with the **-keep_capacitive_coupling** option.

-triplet_type min | typ | max

Specifies which of three values to read from SPEF files (this option does not apply to other formats). Several values in SPEF files, such as capacitor and resistor values, can be specified as triplets (minimum:typical:maximum). By default, NanoTime takes the maximum value. Using this option, you can choose any of the three values.

-dpf_scale_factor *s_factor*

Specifies a factor to apply to the geometrical parameters of DPF transistors. It must be a positive floating-point number and the default is 1.0.

file_names

Specifies the names of one or more parasitic data files.

DESCRIPTION

The **read_parasitics** command reads parasitic data from a file and annotates the information on nets in the current design, thereby allowing more accurate delay calculation to be performed.

The **read_parasitics** command can read SPEF and DSPF parasitic data files. SPEF and DSPF files can be in plain ASCII format or compressed with gzip.

If the **-format** option is not used, NanoTime attempts to read the file type from the header. It is good practice to always specify the format explicitly. If the **-format** option is missing and NanoTime cannot determine the file type, the tool issues an error message.

Net and instance pin names in the design must match instance names in the parasitic data file. For example, if you create the parasitic data file from a design using SPICE naming conventions, the design name must also use SPICE naming conventions.

Resistors and Capacitors in Parasitic Data

The parasitic data can be in either reduced or detailed form. Reduced parasitic data uses an RC pi model for each driver pin of a net; the model includes two capacitors to ground and one resistor between them. Detailed parasitic data consists of a network of resistors, capacitors, and subnodes for each net.

Each capacitor must be connected from a net or subnode to ground. NanoTime splits any cross-coupling capacitor into two separate capacitors, each connected to ground. Each resistor must be connected from one net or subnode to another. NanoTime ignores any resistor that is connected to ground.

NanoTime can handle multiple resistors and capacitors between the same nodes. The tool adds the values of multiple capacitors connected to a node. Multiple resistors between the same two nodes are kept separate and are considered during delay calculation.

NanoTime does not limit the number of resistors and capacitors that can be annotated. However, you can use the **parasitics_warning_net_size** variable to monitor complex annotations that might cause unexpectedly large runtimes. The variable specifies a threshold number of nodes annotated on any single net that should trigger a PARA-003 warning to indicate a large amount of parasitic data on the net.

Completion of Partially Annotated Nets

The **-complete_with** option causes NanoTime to complete the connection from a hierarchical pin to any additional pins at a lower level when the parasitic data is not completely specified for the net. This allows an incomplete parasitic data file to be used. In the absence of this option, NanoTime discards any incomplete RC networks.

The completion algorithm considers the net connected to each hierarchical pin. If all the pins at the next lower hierarchical level are leaf-level transistor pins, NanoTime completes the partial net parasitics by inserting capacitors and resistors between the hierarchical pin and the leaf-level transistor pins.

The argument can take a value of **zero** to insert capacitors and resistors of near-zero value, or **rc** to insert elements with user-defined values.

Reporting Annotated Nets

For each execution of the **read_parasitics** command, NanoTime automatically generates an annotation report after reading the parasitics file by executing the **report_annotated_parasitics -check** command. The tool reports the number of annotated nets, verifies the completeness of annotated RC networks on nets, and checks for dangling RC elements. You can suppress this report by using the **-quiet** option with the **read_parasitics** command. Note that this automatically-generated report includes layout-only nets.

However, if you execute the **report_annotated_parasitics** command separately after executing one or more **read_parasitics** commands, the annotation report provides statistics after the network has been reduced. The report does not include layout-only nets. For these reasons, the results in annotation reports generated at different points in the flow might be different.

To remove parasitics that were read in and annotated with the **read_parasitics** command, use the **remove_annotated_parasitics** command. The **reset_design** command removes all attributes from the current design, including annotated parasitics.

COMMAND PHASING

The **read_parasitics** command can be executed any time after the **link_design** command and before the **check_design** command.

EXAMPLES

The following command checks the parasitic data file adder.spef for syntax errors, but does not annotate the current design.

```
nt_shell> read_parasitics -syntax_only -format SPEF adder.spef
```

The following command reads the parasitics file adder.spef, annotates the current design, and generates an annotation report.

```
nt_shell> read_parasitics -format SPEF adder.spef

Reading file '/.../adder.spef'
****************************************
Report : read_parasitics /.../adder.spef
 ...
****************************************

        0 error(s)
        Format is SPEF
        Annotated nets                  :         7460
        Annotated capacitances          :        86165
        Annotated resistances           :        78705

****************************************
Report : report_annotated_parasitics
        -check
        -internal_nets
        -boundary_nets
        -list_annotated_with_lumped_capacitance
 ...
****************************************
```

```
                                                 Lumped        Not
Net Type                Total RC network  Capacitance  Annotated
------------------ ---------- ---------- ------------ ----------
Internal nets            7252       7250            1          2
Boundary/port nets        210        210            0          0
-------------------------------------------------------------------
                         7462       7460            1          2
1
```

In case a SPEF file contains ports that are appear under multiple names (split), use the following utility to merge all obejcts that belong to the same port. The following example specifies a split-port indicated by a port name followed by an underscore and an integer. abc_0 abc_1 abc_2 will be merged into a single port by the name abc.

```
nt_shell> read_parasitics -format SPEF [merge_spef_split_ports -file adder.spef -regexp {_[0-9]+$}]

The following command removes annotated parasitics from the current design.

nt_shell> remove_annotated_parasitics
1
```

The following set of commands reads in separate parasitics files for several blocks and one file for the top-level design, then checks the annotation completeness.

```
nt_shell> read parasitics -path [all_instances -hierarchy BLKA]
nt_shell> read parasitics -format SPEF B.spef
nt_shell> read parasitics -format SPEF C.spef
nt_shell> read parasitics -format SPEF full_chip.spef
nt_shell> report_annotated_parasitics -check
1
```

The following Tcl command demonstrates the use of the **-path** option to locate parasitics files. The loop cycles through all cells that are instances of the library cell specified by the user variable $cellname. The parasitics files all have the same file name, specified by the user-defined variable $cellspef, but they reside in different directories based on the instance names. The **-path** option argument is the full hierarchical name of the cell instance; NanoTime obtains the name from the cell attribute **full_name**.

If signal integrity (SI) analysis is enabled, NanoTime reads the parasitics files with the **-keep_capacitive_coupling** option; otherwise, that option is not used.

```
foreach_in_collection cell [get_cells -quiet -hierarchy \
                            -filter "ref_name==$cellname"]{
  if {si_enable_analysis == 1} {
    read_parasitics -quiet -increment -keep_capacitive_coupling \
                    -format SPEF \
                    -path [get_attribute $cell full_name] $cellspef
  } else {
    read_parasitics -quiet -increment -format SPEF \
                    -path [get_attribute $cell full_name] $cellspef
  }
}
```

SEE ALSO

```
remove_annotated_parasitics(2)
report_annotated_parasitics(2)
```

```
reset_design(2)
write_parasitics(2)
parasitics_allow_spf_net_override(3)
parasitics_completion_capacitance(3)
parasitics_completion_resistance(3)
parasitics_accept_node_name_net_name(3)
```

read_pattern

Reads in a netlist description for pattern recognition.

SYNTAX

```
status read_pattern
   [-format netlist_type]
   filename
```

Data Types

```
   netlist_type       string
   filename           string
```

ARGUMENTS

-format *netlist_type*

Specifies the format of the pattern netlist: epic, spice, or verilog. If no format is specified, the file is assumed to be in SPICE format.

filename

Specifies the file name of the pattern netlist.

DESCRIPTION

NanoTime can search for occurrences of circuit patterns in the design and perform operations on them. The **read_pattern** command reads in a pattern netlist for use in searching. All subcircuits in the pattern file are saved into the pattern database. NanoTime looks for the specified pattern file in the directories specified by the **search_path** variable.

To report the names of patterns that have been read into NanoTime, use the **list_patterns** command. To search for patterns in the design and perform operations on them, use the **foreach_match** command with the **-pattern** option. The **read_pattern** command must be run after the **link_design** command and

before the **foreach_match** command.

The pattern file can be in the form of a SPICE subcircuit file, such as the following example:

```
*****************************************
* Block: pre_an2k                       *
*****************************************
.subckt pre_an2k clk d1 d2 z
mp1 vdd clk enode vdd p
mp2 vdd z enode vdd p
mp3 vdd clk n1 vdd p
mp4 vdd enode z vdd p
mn1 enode d1 n1 gnd n
mn2 n1 d2 n2 gnd n
mn3 n2 clk gnd gnd n
mn4 z enode gnd gnd n
.ends pre_an2k
```

A SPICE pattern file can contain device parameter information such as transistor widths and lengths, but that information is not used for pattern matching. NanoTime checks for an identical configuration of interconnected NMOS and PMOS transistors, irrespective of transistor parameters other than the NMOS or PMOS type.

COMMAND PHASING

The **read_pattern** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example reads the pattern file named pattern.sp and lists the names of the subcircuits that have been read in as patterns. Then, for each matching occurrence of the pattern in the current design, it applies an **erase_turnoff** command at output net z. The NanoTime tool returns a 1 for every successful application of the command.

```
nt_shell> read_pattern pattern.sp
Compiling "/node1/user1/designs/pattern.sp"
1
nt_shell> list_patterns
Pattern Registry

Pattern Name
----------------------------------------
pre_an2k
1
nt_shell> foreach_match -pattern pre_an2k \
          -command {erase_turnoff -output_net z}
1
1
1
```

SEE ALSO

```
foreach_match(2)
list_patterns(2)
remove_pattern(2)
pattern_merge_parallel_transistors(3)
topo_auto_recognize_user_pattern_command_file(3)
```

read_spice_model

Reads SPICE transistor model definitions into a technology.

SYNTAX

```
status read_spice_model
   -name tech_name
   model_names
```

Data Types

```
   tech_name       string
   model_names     list
```

ARGUMENTS

-name *tech_name*

Specifies the name of the technology to receive the SPICE model definitions. If no technology library name is specified, NanoTime generates a name based on the file name and possibly the operating conditions (temperature and voltage).

model_names

Specifies the names of one or more files containing the SPICE transistor model information.

DESCRIPTION

The **read_spice_model** command reads transistor model information from one or more SPICE files and adds that information to a technology. The specified files must be in plain ASCII format and must contain at least one .lib statement. They must also contain SPICE model file directives.

Unlike the **register_netlist** command, which registers data files for reading later with the **link_design** command, the **read_spice_model** command reads the SPICE file immediately. The command can be run any time before the **check_design** command.

Reading of SPICE transistor models occurs implicitly, without using the **read_spice_model** command, if the SPICE netlist read with the **register_netlist** command and linked by the **link_design** command contains a .lib statement referencing the SPICE models. During linking, NanoTime creates the transistor models and links the transistors in the design to the new models if the **link_enable_netlist_spice_model_linking** variable is set to **true** (the default). Otherwise, model creation and transistor linking occur at the **check_design** command.

You can use the **list_technology** and **report_technology** commands to get information about the transistor models and sizes that have been used and modeled. To remove SPICE models previously read in or to remove technology data lookup tables created from these SPICE models, use the **remove_technology** command.

To prevent the **link_design** command from reading transistor models into technology data lookup tables, set the **link_enable_netlist_spice_model_linking** variable to **false**. In that case, NanoTime does not generate any new models with subsequent **link_design** commands. However, the tool can still read models with the **read_spice_model** commands and can still use transistor models generated while the variable was set to **true**.

If the SPICE technology file contains a SPICE wrapper or macro model, it must be read with the **read_spice_model** command as well as specified as a SPICE netlist with the **register_netlist** command. The **set_technology** command must be executed after the **link_design** command to associate the technology name with the transistors.

SPICE Model File Directives

The SPICE model file contains directives to specify how NanoTime expands the model to make the lookup tables. At a minimum, the SPICE file must contain the following types of statements:

```
*nanosim tech= "delta_vt delta_vtn delta_vtp"
*nanosim tech= "voltage voltage1 voltage2 ..."
*nanosim tech= "vds 0 vdsend vdsstep"
*nanosim tech= "vgs 0 vgsend vgsstep"
*nanosim tech= "body_bias 0 vbsend vbsstep"
.temp 85
.lib '.../libs/model.40' SS
```

Note: For backward compatibility, the following forms of syntax in the SPICE model file are accepted by NanoTime as equivalent directives:

```
*nanosim tech= "directive"
*epic tech= "directive"
```

For more information about the SPICE model directives, see the NanoTime User Guide.

COMMAND PHASING

The **read_spice_model** command can be executed any time before the **check_design** command.

EXAMPLES

The following example reads in a SPICE model file named tech.013um and assigns the models to a

technology named best_case.

```
nt_shell> read_spice_model -name best_case tech.013um
```

SEE ALSO

```
read_techfile(2)
remove_technology(2)
report_technology(2)
list_technology(2)
set_technology(2)
link_enable_corner_specific_wrapper_subckts(3)
link_enable_netlist_spice_model_linking(3)
tech_default_voltage(3)
```

read_techfile

Reads in a Gentech-generated techfile to be used for simulation.

SYNTAX

```
status read_techfile
   [-name name_string]
   file_name
```

Data Types

```
   name_string      string
   file_name        string
```

ARGUMENTS

-name *name_string*

Specifies the name of the technology library to receive the technology model data. If no technology library name is specified, NanoTime generates a name based on the name of the techfile.

file_name

Specifies the name of the Gentech-generated techfile.

DESCRIPTION

The **read_techfile** command reads in transistor model information from a Gentech-generated technology file. This can be done at any time before the **check_design** command. The file can be a plain ASCII text file or an ASCII file that has been compressed with gzip.

You can optionally use the **-name** option to specify the name of a technology library in which to store the technology information. Otherwise, NanoTime uses the name of the technology file as the technology name.

Using multiple **read_techfile** commands, you can read multiple files into the same or different technologies.

You can read in BSIM3/BSIM4 SPICE transistor models (for example, using the **read_spice_model** command) instead of using the **read_techfile** command. In that case, NanoTime creates the technology data lookup tables internally when you use the **link_design** or **set_technology** command. You can use both technology file and SPICE models to get a mixture of techfile-based models and BSIM3/BSIM4 models.

To list the technology data that has been loaded into memory, use the **list_technology** and **report_technology** commands. To remove technology data that has been read in, use the **remove_technology** command.

Before you can analyze the design, each transistor must have a linked model. Linking occurs when you use the **set_technology** command. By default, there must be an exact match between the actual transistor parameters and the techfile-defined model transistor parameters.

You can optionally specify an allowable difference between transistor parameter values in the design and the parameter values of the linked models. To do so, set the following variables:

```
tech_match_length_pct
tech_match_width_pct
tech_match_param_pct
```

Each variable specifies a threshold for mapping transistors to available techfile-defined transistor models. All three parameters are set to zero by default. Set the parameters to positive nonzero values to specify the percentage of allowable difference. For example, set the **tech_match_length_pct** variable to a value of 2.0 to allow the model transistor 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.

COMMAND PHASING

The **read_techfile** command is not phase-restricted.

EXAMPLES

The following command reads in a technology file named pmos.tech:

```
nt_shell> read_techfile pmos.tech
```

The following command reads in a technology file named nmos.tech and puts the transistor models into a technology named best_case:

```
nt_shell> read_techfile -name best_case
```

The following command removes the technology file named nmos.tech in memory:

```
nt_shell> remove_technology -techfile nmos.tech
```

SEE ALSO

list_technology(2)
remove_technology(2)
read_spice_model(2)
report_technology(2)
set_technology(2)
tech_match_length_pct(3)
tech_match_width_pct(3)
tech_match_param_pct(3)

read_topology_library

Reads a user topology library, thereby allowing NanoTime to search for the topologies defined in the library.

SYNTAX

```
status read_topology_library
   [-check]
   directory
```

Data Types

```
   directory        string
```

ARGUMENTS

-check

Checks the specified library directory for errors without actually reading in the library.

directory

The name of the root directory containing the topology library files.

DESCRIPTION

The **read_topology_library** command reads in a user topology library, thereby allowing NanoTime to search for the topologies defined in the library. To enable searching, you must also put the library name in the **topo_library_order** variable. You might also need to enable searching of the individual topologies with the **set_search_enabled** command.

NanoTime reads topology libraries from the directories specified by the **search_path** variable.

The common and global libraries are read in automatically, so you do not need to use the **read_topology_library** command to read them in.

A topology library is a directory containing a set of topology definitions. The directory must contain one file named *dir_name*.ntlib, plus one or two additional files for each defined topology, *topology_name*.libt, and also a *topology_name*.sp file if the search is by pattern and not by subcircuit name.

Using the **-check** option causes NanoTime to check the specified library directory for errors without actually reading in the library. This should be done only after the netlist has been linked. The command checks for existing duplicate lib_topology names, disables searching of subcircuits that do not exist in the design, and verifies the consistency of the pattern name arguments in a multiple-search pattern file.

COMMAND PHASING

The **read_topology_library** command cannot be used before the **link_design** command. To enable the use of a user topology library, you must use the **read_topology_library** command and add the library name to the **topo_library_order** variable before executing any topology recognition commands.

The commands in lib_topology objects might affect the clock network or topology recognition. For this reason, if you use this command after the **match_topology** command, you might need to force another clock propagation and topology recognition operation. To do this, set the **topo_match_topology_reset_clock_and_topology** variable to **true** and execute the **match_topology -force** command to cause NanoTime to repropagate the clock network and rerecognize topologies.

EXAMPLES

The following command reads in the topology libraries named user_lib1 and user_lib2.

```
nt_shell> read_topology_library \
          {user_lib1 user_lib2}
```

The following command checks the topology library named user_lib3 for errors, without actually reading the library into NanoTime.

```
nt_shell> read_topology_library \
          -check user_lib3
```

SEE ALSO

match_topology(2)
check_topology(2)
create_topology_library(2)
list_topology_library(2)
set_search_enabled(2)
remove_search_enabled(2)
search_path(3)
topo_library_order(3)

redirect

Redirects the output of a command to a file.

SYNTAX

```
status redirect
   [-append]
   [-tee]
   [-file | -variable | -channel]
   [-compress]
   [-bg]
   target
   {command_string}
```

Data Types

target	string
command_string	string

ARGUMENTS

-append

Appends the output to the *target* argument.

-tee

Like the UNIX command of the same name, sends output to the current output channel as well as to the *target* argument.

-file

Indicates that the *target* argument is a file name, and redirection is to that file. This is the default. It is exclusive of the **-variable** and **-channel** options.

-variable

Indicates that the *target* argument is a variable name, and redirection is to that Tcl variable. It is exclusive of the **-file** and **-channel** options.

-channel

Indicates that the *target* argument is a Tcl channel, and redirection is to that channel. It is exclusive of the **-variable** and **-file** options.

-compress

Compresses when writing to file. If redirecting to a file, this option specifies that the output should be compressed as it is written. The output file is in a format recognizable by "gzip -d". You cannot specify this option with the **-append** option.

-bg

Indicates that the redirected command executes in the background and in parallel with other foreground commands downstream. NanoTime returns immediately to execute the next command after this redirect command, unless the next command is **exit** or **quit**, which blocks the run until all spawned processes are finished. The **-bg** option cannot be used with any other options except the **-file** option.

target

Indicates the target of the output redirection. This is either a file name, Tcl variable, or Tcl channel depending on the command arguments.

command_string

The command or commands to execute. Intermediate output from each command and the results of the command are redirected to the *target* argument. The *command_string* argument must be rigidly quoted with curly braces.

DESCRIPTION

The **redirect** command performs the same function as the traditional UNIX-style redirection operators > and >>. The *command_string* argument must be rigidly quoted (that is, enclosed in curly braces) in order for the operation to succeed.

Output is redirected to a file by default. Output can be redirected to a Tcl variable by using the **-variable** option, or to a Tcl channel by using the **-channel** option.

Output can be sent to the current output device, as well as the redirect target by using the **-tee** option. See the Examples section for an example.

The **redirect** command is much more flexible than traditional Unix redirection operators. With the **redirect** command, you can redirect multiple commands or an entire script. See the Examples section for an example of how to construct such a command. Do not use the > or >> redirection operators inside the **redirect** command.

Note that the built-in **puts** Tcl command does not respond to output redirection of any kind. Use the built-in **echo** command instead.

The **-bg** option enables parallel processing on multicore systems. NanoTime returns immediately to execute the next command after the **redirect** command, unless the next command is **exit** or **quit**, which blocks the process until all spawned processes have finished. Specifying a file name with the **-bg** option is required; the **-file** option can also be used, but it is optional because it is the default. No other options are allowed. Background processes do not require additional NanoTime licenses.

Each redirect command with the **-bg** option causes NanoTime to spawn a separate process to execute the command string. The results from the command string are placed in the file with the target file name. For each spawned process, a file named by appending ".proc" to the target file name contains information about the spawned process.

The **-bg** option of the **redirect** command is intended only for reporting commands. Use the **parallel_execute** command to improve performance by creating lengthy reports in parallel.

EXAMPLES

In the following example, the output of the plus procedure is redirected. The echoed string and the result of the plus operation is in the output file. Notice that the result is not echoed to the screen.

```
prompt> proc plus {a b} {echo "In plus" ; return [expr $a + $b]}
prompt> redirect p.out {plus 12 13}
prompt> exec cat p.out
In plus
25
```

In this example, a typo in the command creates an error condition. The error message indicates that you can use the **error_info** command to trace the error, but you should first check the output file.

```
prompt> redirect p.out {plus2 12 13}
Error: Errors detected during redirect
        Use error_info for more info. (CMD-013)
prompt> exec cat p.out
Error: unknown command 'plus2' (CMD-005)
```

This example explores the use of results from redirected commands. Since the result of the **redirect** command for a command which does not generate a Tcl error is the empty string, use the **set** command to trap the result of the command. For example, assume that there is a command to read a file which has a result of "1" if it succeeds and "0" if it fails. If you redirect only the command, there is no way to know if it succeeded.

```
redirect p.out { read_a_file "a.txt" }
# Now what?  How can I redirect and use the result?
```

If you set a variable to the result, it is possible to use that result in a conditional expression.

```
redirect p.out { set rres [read_a_file "a.txt"] }
if { $rres == 1 } {
  echo "Read ok!"
}
```

The **redirect** command is not limited to redirection of a single command. You can redirect entire blocks of a script with a single **redirect** command. This example with the **echo** command demonstrates this feature:

```
prompt> redirect p.out {
?         echo -n "Hello "
?         echo "world"
?       }
prompt> exec cat p.out
Hello world
```

```
prompt>
```

The **redirect** command allows you to tee output to the previous output device and also to redirect output to a variable. This example with the **echo** command demonstrates these features:

```
prompt> set y "This is "
This is
prompt> redirect -tee x.out {
          echo XXX
          redirect -variable y -append {
            echo YYY
            redirect -tee -variable z {
              echo ZZZ
            }
          }
        }
XXX
prompt> exec cat x.out
XXX
prompt> echo $y
This is YYY
ZZZ

prompt> echo $z
ZZZ
```

SEE ALSO

```
echo(2)
error_info(2)
set(2)
parallel_execute(2)
```

register_netlist

Specifies one or more netlist files for analysis.

SYNTAX

```
status register_netlist
   [-format netlist_type]
   file_names
```

Data Types

```
   netlist_type       string
   file_names         string
```

ARGUMENTS

-format *netlist_type*

Specifies the format of the netlist file to be registered: epic, spice, or verilog.

file_names

Specifies one or more netlist files to be registered.

DESCRIPTION

The **register_netlist** command specifies the netlist files that contain the design information for analysis. NanoTime checks for the existence of the files and records the file names in the registry. However, the tool does not read the files or check their contents. The netlist files are not actually used until the **link_design** command is executed.

The **-format** option specifies the format of the netlist file to be registered: **epic**, **spice**, or **verilog**. If you do not use the **-format** option, the tool attempts to recognize formats based on the file name extension: .sp for SPICE, .v for Verilog, and so on.

Each netlist can be a plain ASCII text file or an ASCII file that has been compressed with gzip. Compressed files must have the file name extension .gz.

Each file can be specified as an absolute path (starting with a slash character) or a relative path. When you specify a relative path name, NanoTime searches for each file in each directory listed in the **search_path** variable and uses the first one found. To find out the path for a particular relative path name, use the **which** command. When you specify an absolute path name in the **register_netlist** command, NanoTime ignores the **search_path** setting. You can use wildcard characters to specify multiple files.

You can register multiple files with different formats. The **link_design** command automatically translates these different formats into a single internal representation in NanoTime. To use a timing model for timing analysis in place of a netlist, set one of the following variables:

link_prefer_model
link_prefer_model_port

To view the netlist registry, use the **list_netlists** command. To remove netlist files in registry, use the **remove_netlist** command.

COMMAND PHASING

The **register_netlist** command is not phase-restricted.

EXAMPLES

The following **register_netlist** command registers two SPICE netlist files, nand.sp and top.sp. The **search_path** variable is set to a directory called "designs" where the two netlist files are located. The **list_netlists** command lists the files currently in the registry.

```
nt_shell> set search_path "designs"
designs
nt_shell> register_netlist -format spice \
          {nand.sp top.sp}
1
nt_shell> list_netlists
Netlist Registry:

     Format     File File Name
-------------------------------------------------
     spice      designs/nand.sp
     spice      designs/top.sp
1
```

The following example registers two files, a SPICE file and a Verilog file.

```
nt_shell> register_netlist -format spice nand.sp
nt_shell> register_netlist -format verilog top.v
```

The following example removes a netlist file named "top.sp" from the netlist registry.

```
nt_shell> register_netlist -format spice top.sp
nt_shell> register_netlist -format epic nand.ntl
```

```
1
nt_shell> list_netlists
Netlist Registry:

     Format     File File Name
-------------------------------------------------
     spice      designs/top.sp
     epic       designs/nand.ntl
1
nt_shell> remove_netlist */top.sp
nt_shell> list_netlists
Netlist Registry:

     Format     File File Name
-------------------------------------------------
     epic       designs/nand.ntl
1
```

The following example directs NanoTime to use the "inv" timing model from the "inv.db" file. The netlist file "top.sp" contains a subcircuit definition for the design "inv". By default, the subcircuit definition is used rather than the timing model. However, setting the **link_prefer_model** variable to "inv" tells NanoTime to use "inv" timing model instead of the netlist.

```
nt_shell> register_netlist -format spice top.sp
nt_shell> read_library inv.db
nt_shell> set link_prefer_model "inv"
```

SEE ALSO

list_netlists(2)
remove_netlist(2)
list_designs(2)
link_prefer_model(3)
link_prefer_model_port(3)
search_path(3)

remove_allow_pin_swap

Removes an allowance for pins to be swapped in back annotated parasitics.

SYNTAX

```
status remove_allow_pin_swap
   -cell cell_name
   -pins pin_list
```

Data Types

```
   cell        string
   pin_list       list
```

ARGUMENTS

cell_name

The name of the cell or primitive which was previously allowed to have swapped pins in back annotated parasitics.

pin_list

A list of exactly 2 pins that were previously allowed to be swapped in back annotated parasitics.

DESCRIPTION

The **remove_allow_pin_swap** command undoes the effect of the **set_allow_pin_swap** command.

To report currently allowed pin swaps, use **report_allow_pin_swap**.

COMMAND PHASING

The **remove_allow_pin_swap** command is used after the **link_design** command and before the **check_design** command. To be effective, it must be issued prior to the **read_parasitics** command.

RESTRICTIONS

EXAMPLES

The following command defines the two pins t1 and t2 of the rcdel cell to be swappable.

```
nt_shell> set_allow_pin_swap -cell rcdel -pins {t1 t2}
nt_shell> remove_allow_pin_swap -cell rcdel -pins {t1 t2}
```

SEE ALSO

report_allow_pin_swap(2)
remove_allow_pin_swap(2)

remove_annotated_device_parameters

Removes device parameter data previously annotated with the **read_device_parameters** command.

SYNTAX

```
status remove_annotated_device_parameters
   [-all |  cell_list]
```

Data Types

```
   cell_list       list
```

ARGUMENTS

-all

Removes annotated device parameters from all transistors (cells) in the design. This is the default. The **-all** and *cell_list* options are mutually exclusive.

cell_list

A list of transistor cells from which to remove annotated device parameters.

DESCRIPTION

The **remove_annotated_device_parameters** command removes device parameter data previously annotated with the **read_device_parameters** and **read_parasitics** commands. By default, the command removes all annotated device parameter data. To remove device parameter data from specified transistor cells, list the cells in the command.

COMMAND PHASING

The **remove_annotated_device_parameters** command is typically executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design**

commands). If you use the **remove_annotated_device_parameters** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes all annotated device parameter data from the design.

```
nt_shell> remove_annotated_device_parameters
```

The following command removes annotated device parameter data from transistor cells x1 and x2.

```
nt_shell> remove_annotated_device_parameters {x1 x2}
```

The following command removes annotated device parameter data from all cells whose names begin with "Xn".

```
nt_shell> remove_annotated_device_parameters [get_cells Xn*]
```

SEE ALSO

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

remove_annotated_parasitics

Removes parasitics from the design.

SYNTAX

```
status remove_annotated_parasitics
   [-all]
   [-capacitance]
   [nets]
```

Data Types

```
   nets         list
```

ARGUMENTS

-all

Removes annotated parasitics from all nets in the design. This is the default. The **-all** and *net_list* options are mutually exclusive.

-capacitance

Removes only the annotated capacitances. If this option is not used, capacitances and resistances are both removed.

nets

A list of nets from which to remove annotated parasitics.

DESCRIPTION

The **remove_annotated_parasitics** command removes all parasitics from the design, including parasitics read in by using the **read_parasitics** command and parasitics embedded in the design netlist.

By default, the command removes all annotated parasitic data. To remove parasitic data from specified

nets, list the nets in the argument list. To remove only capacitances, use the **-capacitance** option.

COMMAND PHASING

The **remove_annotated_parasitics** command is typically executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **remove_annotated_parasitics** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes all annotated parasitic data from the design.

```
nt_shell> remove_annotated_parasitics
```

The following command removes annotated parasitic data from nets n1, n2, and n3.

```
nt_shell> remove_annotated_parasitics {n1 n2 n3}
```

The following command removes annotated parasitic data from all nets whose names begin with "Sn".

```
nt_shell> remove_annotated_parasitics [get_nets Sn*]
```

SEE ALSO

read_parasitics(2)
report_annotated_parasitics(2)

remove_capacitance

Removes capacitance previously set on ports and nets with the **set_load** command.

SYNTAX

```
status remove_capacitance
   port_or_net_list
```

Data Types

```
   port_or_net_list       list
```

ARGUMENTS

port_or_net_list

The list of ports and nets from which to remove capacitance values.

DESCRIPTION

The **remove_capacitance** command removes the capacitance previously set on specified nets or ports with the **set_load** command. NanoTime reverts to using the capacitance from detailed parasitics (applied with the **read_parasitics** command), if any.

COMMAND PHASING

The **remove_capacitance** command is typically executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **remove_capacitance** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example removes the capacitance on port ox6 and net n12.

```
nt_shell> remove_capacitance [get_ports ox6]
nt_shell> remove_capacitance [get_nets n2]
```

SEE ALSO

read_parasitics(2)
set_load(2)

remove_case_analysis

Removes case analysis values previously set with the **set_case_analysis** command.

SYNTAX

```
status remove_case_analysis port_or_pin_list
```

Data Types

```
port_or_pin_list    list
```

ARGUMENTS

port_or_pin_list

The ports or pins with previously defined case analysis settings that you want to remove.

DESCRIPTION

The **remove_case_analysis** command removes case analysis values previously set with the **set_case_analysis** command. The command must specify one or more ports or pins from which to remove case analysis values.

When the case analysis value is removed, all the constant values propagated from the case value are removed, which include the values propagated through the transistors and the combinational cell functions.

Setting or removing logic values changes the design, making it necessary to run the **check_design** and **trace_paths** commands to get new analysis results.

COMMAND PHASING

The **remove_case_analysis** command is typically executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **remove_case_analysis** command after the **check_design** command, NanoTime

transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example removes the case analysis setting on the port reset_in.

```
nt_shell> remove_case_analysis
          [get_ports reset_in]
```

SEE ALSO

set_case_analysis(2)
report_case_analysis(2)

remove_clock

Removes a clock previously created by the **create_clock** command.

SYNTAX

```
status remove_clock
   [-all]
   [clock_list]
```

Data Types

```
   clock_list        list
```

ARGUMENTS

-all

Removes all clocks previously created by the **create_clock** command.

clock_list

The list of clocks to be removed. It can be a list of one or more clock names or patterns that match clock names, or a list of collections containing clocks.

DESCRIPTION

Removes the specified clock objects previously created by the **create_clock** command from the current design. You must specify either the **-all** option to remove all clocks or a list of clocks to remove.

All timing constraints that are set relative to the clock being removed (for example, using the **set_input_delay** and **set_output_delay** commands) are also removed.

To show a list all clocks currently defined in the design, use the **report_clock** command.

If you use the **remove_clock** command in a topology library object, it should be used with the **-**

action_pre_match_topology option in the **create_lib_topology** command.

COMMAND PHASING

The **remove_clock** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_clock** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

If you use the **remove_clock** command after a **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is **true**, NanoTime repropagates the clock network and rerecognizes topologies at every subsequent **match_topology -force** command.

EXAMPLES

The following command removes clock CLK1 from the design, together with all constraints set relative to that clock.

```
nt_shell> remove_clock CLK1
```

The following command removes all clocks that begin with the characters CL.

```
nt_shell> remove_clock CL*
```

The following command removes all clocks from the current design.

```
nt_shell> remove_clock -all
```

SEE ALSO

```
create_clock(2)
create_lib_topology(2)
current_design(2)
get_clocks(2)
report_clock(2)
set_input_delay(2)
set_output_delay(2)
reset_design(2)
topo_match_topology_reset_clock_and_topology(3)
```

remove_clock_latency

Removes clock latency previously set with the **set_clock_latency** command from specified objects.

SYNTAX

```
status remove_clock_latency
   [-source]
   object_list
```

Data Types

```
   object_list       list
```

ARGUMENTS

-source

Removes source latency (rather than network latency) from the specified object.

object_list

Specifies the clocks, ports, or pins from which to remove the latency value.

DESCRIPTION

The **remove_clock_latency** command removes latency values previously set with the **set_clock_latency** command from specified clocks, ports, or pins.

A clock or port can have a both source latency and network latency values. Use the **-source** option to remove the source latency, or omit the **-source** option to remove the network latency.

Setting network latency on a propagated clock converts that clock to ideal. Removing network latency does not automatically change the clock back to propagated. To make the clock propagated, use the **set_propagated_clock** command.

For more information about clock latency, see the man page for the **set_clock_latency** command.

To report network or source latency set on a clock, use the **report_clock** command with the **-skew** option.

To report network or source latency set on a port or pin, use the **get_attribute** command to examine relevant port or pin attributes.

COMMAND PHASING

The **remove_clock_latency** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_clock_latency** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example removes clock latency information from clock CLK1.

```
nt_shell> remove_clock_latency [get_clocks CLK1]
```

The following example removes clock source latency information from clock CLK1.

```
nt_shell> remove_clock_latency -source [get_clocks CLK1]
```

SEE ALSO

```
create_clock(2)
remove_clock(2)
remove_clock_uncertainty(2)
report_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)
```

remove_clock_transition

Removes clock transition time information previously set with the **set_clock_transition** command.

SYNTAX

```
status remove_clock_transition
   clock_list
```

Data Types

```
   clock_list      list
```

ARGUMENTS

clock_list

Specifies one or more clocks from which to remove the clock transition time.

DESCRIPTION

The **remove_clock_transition** command removes clock transition information previously set by the **set_clock_transition** command.

To list all the set clock transition values, use the **report_clock -skew** command.

COMMAND PHASING

The **remove_clock_transition** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_clock_transition** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example removes clock transition information from all clocks in the current design.

nt_shell> **remove_clock_transition [all_clocks]**

SEE ALSO

all_clocks(2)
report_clock(2)
set_clock_transition(2)
create_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)

remove_clock_uncertainty

Removes clock uncertainty information previously set by the **set_clock_uncertainty** command.

SYNTAX

```
status remove_clock_uncertainty
   [object_list |
    -from from_clock
      | -rise_from rise_from_clock
      | -fall_from fall_from_clock
    -to to_clock
      | -rise_to rise_to_clock
      | -fall_to fall_to_clock]
   [-rise]
   [-fall]
   [-setup]
   [-hold]
   [-same_cycle]
```

Data Types

```
   object_list          list
   from_clock           list
   rise_from_clock      list
   fall_from_clock      list
   to_clock             list
   rise_to_clock        list
   fall_to_clock        list
```

ARGUMENTS

object_list

Specifies a list of clocks, ports, pins, or cells from which to remove uncertainty information.

-from *from_clock*

Specifies the source clock for interclock uncertainty. You can specify a pair of **-from/-rise_from/-fall_from** and **-to/-rise_to/-fall_to** options or an *object_list*, but not both. If you do not specify any options or object lists, uncertainty information is removed from all objects in the current design.

-rise_from *rise_from_clock*

Same as the **-from** option, but indicates that uncertainty removal applies only to rising edges of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-fall_from ***fall_from_clock***

Same as the **-from** option, but indicates that uncertainty removal applies only to falling edges of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-to ***to_clock***

Specifies the destination clock for interclock uncertainty.

-rise_to ***rise_to_clock***

Same as the **-to** option, but indicates that uncertainty removal applies only to rising edges of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-fall_to ***fall_to_clock***

Same as the **-to** option, but indicates that uncertainty removal applies only to falling edges of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise

Specifies that uncertainty is to be removed for only the rising clock edge. By default, uncertainty is removed for both rising and falling clock edges. This option is valid only for interclock uncertainty.

-fall

Specifies that uncertainty is to be removed for only the falling clock edge. By default, uncertainty is removed for both rising and falling clock edges. This option is valid only for interclock uncertainty.

-setup

Removes the uncertainty value only for setup checks. By default, removal applies to both setup and hold checks.

-hold

Removes the uncertainty value only for hold checks. By default, removal applies to both setup and hold checks.

-same_cycle

Removes the uncertainty value only for same-cycle checks. By default, removal applies to both next-cycle and same-cycle checks.

DESCRIPTION

Removes clock uncertainty information previously set by the **set_clock_uncertainty** command from clocks, ports, pin, or cells, or between specified clocks. If the command is issued without options, all clock uncertainty information is removed from the current design.

To view the current clock uncertainty settings, use the **report_clock -skew** command.

COMMAND PHASING

The **remove_clock_uncertainty** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_clock_uncertainty** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example removes uncertainty information from clock CLK and pin clk_buf/Z.

```
nt_shell> remove_clock_uncertainty [get_clocks CLK]
nt_shell> remove_clock_uncertainty [get_pins clk_buf/Z]
```

The following example removes interclock uncertainty within and between the PHI1 and PHI2 clock domains.

```
nt_shell> remove_clock_uncertainty -from PHI1 -to PHI1
nt_shell> remove_clock_uncertainty -from PHI2 -to PHI2
nt_shell> remove_clock_uncertainty -from PHI1 -to PHI2
nt_shell> remove_clock_uncertainty -from PHI2 -to PHI1
```

SEE ALSO

all_clocks(2)
create_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)

remove_conservative_max_delay

Disables nets for conservative (simultaneous switching) maximum delay analysis for multi-input switching analysis.

SYNTAX

```
status remove_conservative_max_delay
   net_list
```

Data Types

net_list	list

ARGUMENTS

net_list

Specifies a list of nets for which conservative max delay analysis should be disabled.

DESCRIPTION

The **remove_conservative_max_delay** command cancels the effects of a previously executed **set_conservative_max_delay** command.

This command disables the conservative maximum delay analysis for the entire channel-connected region represented by each net in the list.

COMMAND PHASING

The **remove_conservative_max_delay** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_conservative_max_delay** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example identifies all the nets whose names match int* as ground nets.

```
nt_shell> remove_conservative_max_delay int*
```

SEE ALSO

```
remove_conservative_min_delay(2)
set_conservative_max_delay(2)
set_conservative_min_delay(2)
```

remove_conservative_min_delay

Disables nets for conservative (simultaneous switching) minimum delay analysis.

SYNTAX

```
status remove_conservative_min_delay
   net_list
```

Data Types

```
   net_list     list
```

ARGUMENTS

net_list

Specifies a list of nets for which conservative minimum delay analysis should be disabled.

DESCRIPTION

The **remove_conservative_min_delay** command cancels the effects of a previously executed **set_conservative_min_delay** command.

This command disables the conservative minimum delay analysis for the entire channel-connected region represented by each net in the list.

COMMAND PHASING

The **remove_conservative_min_delay** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_conservative_min_delay** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example identifies all the nets whose names match int* as ground nets.

```
nt_shell> remove_conservative_min_delay int*
```

SEE ALSO

```
remove_conservative_max_delay(2)
set_conservative_max_delay(2)
set_conservative_min_delay(2)
```

remove_data_check

Removes timing constraints previously set with the **set_data_check** command.

SYNTAX

```
status remove_data_check
   {-from from_object
      | -rise_from from_object
      | -fall_from from_object}
   {-to to_object
      | -rise_to to_object
      | -fall_to to_object}
   [-setup | -hold]
   [-of_objects  timing_check_objects]
```

Data Types

```
   from_object              string
   to_object                string
   timing_check_objects     list
```

ARGUMENTS

-from *from_object*

Specifies a pin or port in the current design as the related pin of the data-to-data check. You must specify one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising delays at the related pin.

-fall_from *from_object*

Similar to the **-from** option, but applies only to falling delays at the related pin.

-to *to_object*

Specifies a pin or port in the current design as the constrained pin of the data-to-data check. You must specify one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***to_object***

Similar to the **-to** option, but applies only to rising delays at the constrained pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling delays at the constrained pin.

-setup

Indicates a setup data check.

-hold

Indicates a hold data check.

-of_objects ***timing_check_objects***

Remove all data checks from this list of timing check objects.

DESCRIPTION

The **remove_data_check** command removes data-to-data checks previously set with the **set_data_check** command. These types of checks are constraints evaluated after all path tracing is complete, including data-data, clock-clock, and pulse-width checks.

In the **remove_data_check** command, you must specify the same options as the original **set_data_check** command, including the "from" and "to" type options and **-setup** and **-hold** options (if used originally).

COMMAND PHASING

The **remove_data_check** command must be executed after the **link_design** command but before the **trace_paths** command.

EXAMPLES

The following command removes a setup check from d[3] rising to result[7] falling. A return value of 1 indicates that at least one constraint was removed. A 0 is returned if no constraints could be removed.

```
nt_shell> remove_data_check -rise_from d[3] \
          -fall_to result[7] -setup
```

SEE ALSO

set_data_check(2)

remove_data_trace_from_clock

Disables data tracing from the startpoints of specified clocks.

SYNTAX

```
status remove_data_trace_from_clock
   clock_list
```

Data Types

```
   clock_list          list
```

ARGUMENTS

clock_list

Specifies a collection of clocks affected by the command. NanoTime does not perform data tracing from the startpoints of the specified clocks.

DESCRIPTION

During path tracing, NanoTime performs data tracing from all user-defined inputs and clock tracing from all clock startpoints. Clock tracing finds each path that starts at the boundary of the design and ends at the clock input of a sequential element such as a latch, precharge structure, or RAM cell.

In addition, by default, NanoTime performs data tracing from all clock startpoints. Data tracing finds each path that starts at a clock port, goes from the clock input to the Q output of a latch, and continues to another latch or to the output of the design.

To prevent NanoTime from performing data tracing for a given clock, use the **remove_data_trace_from_clock** command to specify the clock. If you later want to cancel the effects of this command and enable data tracing of paths for a given clock, use the **set_data_trace_from_clock** command. The default behavior is to perform data tracing for all clocks.

COMMAND PHASING

The **remove_data_trace_from_clock** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **remove_data_trace_from_clock** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command prevents data tracing from clock startpoints of clock ck1:

```
nt_shell> remove_data_trace_from_clock [get_clocks ck1]
1
```

The following command enables data tracing from clock startpoints of clock ck1, thus restoring the default behavior.

```
nt_shell> set_data_trace_from_clock [get_clocks ck1]
1
```

SEE ALSO

set_data_trace_from_clock(2)
trace_paths(2)

remove_dcs_input

Removes the existing constant logic value previously set on a net, port, or pin that acts as a side input for dynamic clock simulation.

SYNTAX

```
status remove_dcs_input
   [-logic]
   [-synchronized_data]
   object_list
```

Data Types

```
   object_list   list
```

ARGUMENTS

-logic

Specifies removal of the existing constant logic value (0 or 1) previously set by the **set_dcs_input** command on a net, port, or pin that acts as a side input for dynamic clock simulation.

-synchronized_data

Specifies removal of signals in self-timed circuits that turn off the sense amplifier enable signals.

object_list

The nets, ports, or pins under consideration.

DESCRIPTION

The **remove_dcs_input** command specifies removal of a constant logic value (0 or 1) previously set with a **set_dcs_input** command on a net, port, or pin.

To change the constant logic value of a side input, you must use the **remove_dcs_input** command

before executing another **set_dcs_input** command on that side input. Otherwise, all **set_dcs_input** commands beyond the first **set_dcs_input** command result in an error.

COMMAND PHASING

The **remove_dcs_input** command must be executed after the **check_topology** command but before the **check_design** command. If you use the **remove_dcs_input** command after the **check_design** command, NanoTime discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example shows the use of the **remove_dcs_input** command to change the constant logic value on net xccarray/nsel0 from logic 0 to logic 1.

```
nt-shell> set_dcs_input -logic 0 xccarray/nsel0
1
nt-shell> remove_dcs_input -logic xccarray/nsel0
1
nt-shell> set_dcs_input -logic 1 xccarray/nsel0
1

nt-shell> check_design
Starting dynamic clock simulation...
Finished dynamic clock simulation
Checking timing graph for consistency...
1
```

SEE ALSO

set_dcs_input(2)

remove_delay_coefficients

Removes delay coefficients previously set with the **set_delay_coefficients** command.

SYNTAX

```
status remove_delay_coefficients
   objects
```

Data Types

```
   objects        list
```

ARGUMENTS

objects

The list of nets or transistor gate pins affected by the command. This list defines a set of delay calculation trigger pins or nets. The delays from these trigger pins or nets to a driven output net is no longer modified.

DESCRIPTION

The **remove_delay_coefficients** command removes delay coefficients previously set with the **set_delay_coefficients** command. You must specify a list of nets or transistor gates from which to remove the coefficients. After running the command, the delays are calculated normally at the specified nets or transistor gates.

To remove transition time coefficients, use the **remove_transition_coefficients** command.

COMMAND PHASING

The **remove_delay_coefficients** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **remove_delay_coefficients** command after the **check_design** command, NanoTime transitions

back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes all delay coefficients previously set on net Sn04.

```
nt_shell> remove_delay_coefficients [get_nets Sn04]
```

SEE ALSO

```
set_delay_coefficients(2)
report_delay_coefficients(2)
remove_transition_coefficients(2)
set_transition_coefficients(2)
```

remove_differential

Removes differential marking from nets, pins, or ports.

SYNTAX

```
status remove_differential
   differential_objects
```

Data Types

```
   differential_objects      list
```

ARGUMENTS

differential_objects

Specifies a list of objects from which the differential marking is to be removed.

DESCRIPTION

This command removes the marking of differential pairs from the objects in the argument list. The objects must have been previously included in one or more **set_differential** commands.

COMMAND PHASING

The **remove_differential** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **remove_differential** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following command removes the differential marking on net in3p, which was previously specified as part of a differential pair with the **set_differential** command.

```
nt_shell> remove_differential  { in3p }
```

SEE ALSO

set_differential(2)

remove_disable_logic_check

Cancels the effects of the **set_disable_logic_check** command for specified nets in the design.

SYNTAX

```
status remove_disable_logic_check
   nets
```

Data Types

```
   nets          list
```

ARGUMENTS

nets

The list of nets that are to have normal logic checking.

DESCRIPTION

The **remove_disable_logic_check** command cancels the effects of a previous **set_disable_logic_check** command for specified nets, which restores normal logic checking for those nets.

COMMAND PHASING

The **remove_disable_logic_check** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_disable_logic_check** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command enables normal logic checking of net pgen1.

```
nt_shell> remove_disable_logic_check pgen1
```

SEE ALSO

set_disable_logic_check(2)

remove_drive_resistance

Removes drive resistance values previously set in input ports and bidirectional ports with the **set_drive** or **set_drive_resistance** command.

SYNTAX

```
status remove_drive_resistance
   port_list
```

Data Types

```
   port_list          list
```

ARGUMENTS

port_list

A list of input and bidirectional ports in the current design from which to remove driver resistance values.

DESCRIPTION

The **remove_drive_resistance** command removes the drive resistance values from one or more input ports or bidirectional ports. It cancels the effects of the **set_drive** or **set_drive_resistance** command.

To find out the drive resistance values set on ports, use the **report_port -drive** command.

COMMAND PHASING

The **remove_drive_resistance** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_drive_resistance** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes the drive resistance values previously set on ports A, B, and C.

```
nt_shell> remove_drive_resistance [get_ports {A B C}]
```

SEE ALSO

report_port(2)
set_drive(2)

remove_driving_cell

Removes port driving cell information.

SYNTAX

```
status remove_driving_cell
   port_list
```

Data Types

```
   port_list      list
```

ARGUMENTS

port_list

Specifies a list of input or output ports.

DESCRIPTION

Removes driving cell information from the specified ports. The driving cell information is specified with the **set_driving_cell** command.

To see port drive information, use the **report_port -drive** command.

COMMAND PHASING

The **remove_driving_cell** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_driving_cell** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example removes all driving cell information for port IN2.

```
nt_shell> remove_driving_cell IN2
```

SEE ALSO

report_port(2)
set_driving_cell(2)

remove_exclude_reference_path

Cancels the effects of a previous **set_exclude_reference_path** command.

SYNTAX

```
status remove_exclude_reference_path
   [-max] [-min]
   [-exclude_reference_through exclude_reference_through_list
      | -exclude_reference_rise_through exclude_reference_rise_through_list
      | -exclude_reference_fall_through exclude_reference_fall_through_list]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

```
   exclude_reference_through_list      list
   exclude_reference_rise_through_list list
   exclude_reference_fall_through_list list
   from_list                           list
   rise_from_list                      list
   fall_from_list                      list
   to_list                             list
   rise_to_list                        list
   fall_to_list                        list
   through_list                        list
   rise_through_list                   list
   fall_through_list                   list
```

ARGUMENTS

-max

Removes an exclude_reference exception previously set with a matching **-max** option.

-min

Removes an exclude_reference exception previously set with a matching **-min** option.

-exclude_reference_through *exclude_reference_through_list*

Removes an exclude_reference exception previously set with a matching **-exclude_reference_through** option.

-exclude_reference_rise_through *exclude_reference_rise_through_list*

Removes an exclude_reference exception previously set with a matching **-exclude_reference_rise_through** option.

-exclude_reference_fall_through *exclude_reference_fall_through_list*

Removes an exclude_reference exception previously set with a matching **-exclude_reference_fall_through** option.

-from *from_list*

Removes an exclude_reference exception previously set with a matching **-from** option.

-rise_from *rise_from_list*

Removes an exclude_reference exception previously set with a matching **-rise_from** option.

-fall_from *fall_from_list*

Removes an exclude_reference exception previously set with a matching **-fall_from** option.

-to *to_list*

Removes an exclude_reference exception previously set with a matching **-to** option.

-rise_to *rise_to_list*

Removes an exclude_reference exception previously set with a matching **-rise_to** option.

-fall_to *fall_to_list*

Removes an exclude_reference exception previously set with a matching **-fall_to** option.

-through *through_list*

Removes an exclude_reference exception previously set with a matching **-through** option.

-rise_through *rise_throughlist*

Removes an exclude_reference exception previously set with a matching **-rise_through** option.

-fall_through *fall_through_list*

Removes an exclude_reference exception previously set with a matching **-fall_through** option.

DESCRIPTION

The **remove_exclude_reference_path** command cancels the effects of a previous **set_exclude_reference_path** command. It removes exclude_reference exception definitions that can be listed with the **report_exceptions** command.

You must use at least one of the **-exclude_reference_through**, **-from**, **-through**, or **-to** type options to specify which exclude_reference exception definitions to remove. A general **remove_exclude_reference_path** command removes the effects of a matching general or more specific **set_exclude_reference_path** command, as long as there is a matching object in the path specification of the original **set_exclude_reference_path** command.

COMMAND PHASING

The **remove_exclude_reference_path** command can be executed any time after the **set_exclude_reference_path** command to which it relates.

EXAMPLES

Suppose that you define the following exclude_reference exceptions:

```
nt_shell> set_exclude_reference_path 2 \
          -from selA -through clka -exclude_reference_through {clkb} -to BUS[60]
nt_shell> set_exclude_reference_path 2 \
          -from selB -through clkb -exclude_reference_through {clka} -to BUS[61]
nt_shell> report_exceptions

From    Through   To         Setup       Hold
-----------------------------------------------------
selA    {clka}
        {clkb(e)} BUS[60]    exclude_ref exclude_ref
selB    {clkb}
        {clka(e)} BUS[61]    exclude_ref exclude_ref
```

To remove the first exclude_reference exception from the list of exceptions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_exclude_reference_path \
          -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

nt_shell> **remove_exclude_reference_path -from selA**

You could remove both sets of selected of paths by using the following command:

nt_shell> **remove_exclude_reference_path -from sel***

This is because the **-from sel*** option matches the object specifications used in the original **set_exclude_reference_path** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_exclude_reference_path \
          -through [get_nets n31]
```

```
nt_shell> remove_exclude_reference_path \
          -rise_from selA
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_exclude_reference_path** command. You cannot remove just the rising-edge path exceptions from selA because that is more specific than the **-from** option used in the original **set_exclude_reference_path** command.

In other words, you cannot remove a subset of exclude_reference exception definitions specified by a single **set_exclude_reference_path** command. You can only remove whole items listed in the **report_exceptions** report.

SEE ALSO

```
report_exceptions(2)
set_exclude_reference_path(2)
```

remove_false_path

Cancels the effects of a previous **set_false_path** command.

SYNTAX

```
status remove_false_path
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

from_list	list
rise_from_list	list
fall_from_list	list
to_list	list
rise_to_list	list
fall_to_list	list
through_list	list
rise_through_list	list
fall_through_list	list

ARGUMENTS

-setup

Removes a false path timing exception previously set with a matching **-setup** option.

-hold

Removes a false path timing exception previously set with a matching **-hold** option.

-rise

Removes a false path timing exception previously set with a matching **-rise** option.

-fall

Removes a false path timing exception previously set with a matching **-fall** option.

-from ***from_list***

Removes a false path timing exception previously set with a matching **-from** option.

-rise_from ***rise_from_list***

Removes a false path timing exception previously set with a matching **-rise_from** option.

-fall_from ***fall_from_list***

Removes a false path timing exception previously set with a matching **-fall_from** option.

-to ***to_list***

Removes a false path timing exception previously set with a matching **-to** option.

-rise_to ***rise_to_list***

Removes a false path timing exception previously set with a matching **-rise_to** option.

-fall_to ***fall_to_list***

Removes a false path timing exception previously set with a matching **-fall_to** option.

-through ***through_list***

Removes a false path timing exception previously set with a matching **-through** option.

-rise_through ***rise_through_list***

Removes a false path timing exception previously set with a matching **-rise_through** option.

-fall_through ***fall_through_list***

Removes a false path timing exception previously set with a matching **-fall_through** option.

DESCRIPTION

The **remove_false_path** command cancels the effects of a previous **set_false_path** command. It removes false path definitions that can be listed with the **report_exceptions** command.

You must use at least one of the **-from**, **-through**, or **-to** type options to specify which false path definitions to remove. A general **remove_false_path** command removes the effects of a matching general or more specific **set_false_path** command, as long as there is a matching object in the path specification of the original **set_false_path** command. For example, suppose that you define the following false paths:

```
nt_shell> set_false_path -from selA -to BUS[60]
nt_shell> set_false_path -from selB -to BUS[61]
```

```
nt_shell> report_exceptions
...
From     Through   To         Setup   Hold     Matches
-----------------------------------------------------
selA     *         BUS[60]    FALSE   FALSE    --
selB     *         BUS[61]    FALSE   FALSE    --
```

To remove the first false path definition from the list of exceptions (from selA to BUS[60]), you could use the following command:

nt_shell> **remove_false_path -from selA -to BUS[60]**

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

nt_shell> **remove_false_path -from selA**

You could remove both sets of selected of paths by using the following command:

nt_shell> **remove_false_path -from sel***

This is because the **-from sel*** option matches the object specifications used in the original **set_false_path** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_false_path -through [get_nets n31]
nt_shell> remove_false_path -rise_from selA
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_false_path** command. You cannot remove just the rising-edge path exceptions from selA because that is more specific than the **-from** option used in the original **set_false_path** command.

In other words, you cannot remove a subset of false path definitions specified by a single **set_false_path** command. You can only remove whole items listed in the **report_exceptions** report.

COMMAND PHASING

The **remove_false_path** command can be executed any time after the **set_false_path** command to which it relates.

EXAMPLES

The following example demonstrates how to selectively remove false path definitions.

```
nt_shell> set_false_path \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_false_path \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
```

```
1
nt_shell> set_false_path \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_exceptions

From       Through   To         Setup   Hold   Matches
-------------------------------------------------------
selA       *         BUS[60]    FALSE   FALSE   --
selB       *         BUS[61]    FALSE   FALSE   --
clk1(r)    *         clk2(f)    FALSE   FALSE   --

1
nt_shell> remove_false_path \
          -from [get_clocks clk1]
1
nt_shell> report_exceptions

From       Through   To         Setup   Hold   Matches
-------------------------------------------------------
selA       *         BUS[60]    FALSE   FALSE   --
selB       *         BUS[61]    FALSE   FALSE   --
```

The **remove_false_path** command above removes the clock-to-clock false path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** option.

```
nt_shell> remove_false_path \
          -from [get_ports selA]

nt_shell> report_exceptions

From       Through   To         Setup   Hold   Matches
-------------------------------------------------------
selB       *         BUS[61]    FALSE   FALSE   --
```

The **remove_false_path** command above removes the false path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

report_exceptions(2)
set_false_path(2)

remove_find_path

Cancels the effects of the **set_find_path** command by removing items from the current list of selected paths.

SYNTAX

```
status remove_find_path
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

from_list	list
rise_from_list	list
fall_from_list	list
to_list	list
rise_to_list	list
fall_to_list	list
through_list	list
rise_through_list	list
fall_through_list	list

ARGUMENTS

-from *from_list*

Removes a find path specification previously set with a matching **-from** option.

-rise_from *rise_from_list*

Removes a find path specification previously set with a matching **-rise_from** option.

-fall_from *fall_from_list*

Removes a find path specification previously set with a matching **-fall_from** option.

-to ***to_list***

Removes a find path specification previously set with a matching **-to** option.

-rise_to ***rise_to_list***

Removes a find path specification previously set with a matching **-rise_to** option.

-fall_to ***fall_to_list***

Removes a find path specification previously set with a matching **-fall_to** option.

-through ***through_list***

Removes a find path specification previously set with a matching **-through** option.

-rise_through ***rise_through_list***

Removes a find path specification previously set with a matching **-rise_through** option.

-fall_through ***fall_through_list***

Removes a find path specification previously set with a matching **-fall_through** option.

DESCRIPTION

The **remove_find_path** command cancels the effects of the **set_find_path** command by removing specified paths from the current sets of paths listed by the **report_find_path** command. The **remove_find_path** and **set_find_path** commands have the same **-from**, **-through**, and **-to** type options.

A general **remove_find_path** command removes the effects of a matching general or more-specific **set_find_path** command, as long as there is a matching object in the path specification of the original **set_find_path** command. For example, suppose that you select the following paths:

```
nt_shell> set_find_path -from selA -to BUS[60]
nt_shell> set_find_path -from selB -to BUS[61]
nt_shell> report_find_path

From               Through            To
-----------------------------------------------
selA               *                  BUS[60]
selB               *                  BUS[61]
```

To remove the first item from the list of path restrictions (from selA to BUS[60]), you could use the following command:

nt_shell> **remove_find_path -from selA -to BUS[60]**

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

nt_shell> **remove_find_path -from selA**

You could remove both sets of selected of paths by using the following command:

nt_shell> **remove_find_path -from sel***

This is because the **-from sel*** option matches the object specifications used in the original **set_find_path** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_find_path -through [get_nets n31]
nt_shell> remove_find_path -rise_from selA
```

You cannot remove the subset of paths passing through net n31 because there is no matching object in the original **set_find_path** command. You cannot remove just the rising-edge paths from selA because that is more specific than the **-from** option used in the original **set_find_path** command.

In other words, you cannot remove a subset of paths specified by a single **set_find_path** command. You can only remove whole items listed in the **report_find_path** report.

COMMAND PHASING

The **remove_find_path** command can be executed any time after the **set_find_path** command to which it relates.

EXAMPLES

The following example demonstrates how to selectively remove paths selected for path tracing.

```
nt_shell> set_find_path \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_find_path \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_find_path \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_find_path

From              Through           To
-----------------------------------------------
selA              *                 BUS[60]
selB              *                 BUS[61]
clk1(r)           *                 clk2(f)

1
nt_shell> remove_find_path \
          -from [get_clocks clk1]
1
nt_shell> report_find_path
...
From              Through           To
-----------------------------------------------
selA              *                 BUS[60]
```

```
selB                *                   BUS[61]
```

The **remove_find_path** command above removes the clock-to-clock path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** option.

```
nt_shell> remove_find_path \
          -from [get_ports selA]

nt_shell> report_find_path

From                Through             To
-------------------------------------------
selB                *                   BUS[61]
```

The **remove_find_path** command above removes the path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

check_design(2)
report_find_path(2)
set_find_path(2)
trace_paths(2)

remove_from_collection

Removes objects from a collection, resulting in a new collection. The base collection remains unchanged.

SYNTAX

```
collection remove_from_collection
   base_collection
   object_spec
   [-intersect]
```

ARGUMENTS

base_collection

Specifies the base collection to be copied to the resulting collection before object removal from the resulting collection.

object_spec

Specifies a list of named objects or collections to compare to the base collection.

-intersect

Keeps object(s) that are found in *object_spec* (the default is to remove them).

DESCRIPTION

The **remove_from_collection** command creates a new collection that consists of elements determined by comparing a base collection and a list of objects. The base collection is unchanged; instead, NanoTime creates a copy of the base collection and removes objects from the resulting collection.

In the absence of the **-intersect** option, objects in the base collection that match objects in *object_spec* are removed from the resulting collection. If the **-intersect** option is present, the matching objects are kept in the resulting collection and all other objects are removed.

The object class of each element in the *object_spec* list must be the same as in the base collection. If an

item in the *object_spec* list matches an existing collection, the collection is used. Otherwise, NanoTime searches for objects in the database which are of the same object class as the base collection.

If the base collection is homogeneous, any element of *object_spec* that is not a collection is searched for. If the base collection is heterogeneous, any element of *object_spec* that is not a collection is ignored.

The resulting collection can be a copy of the base collection or the empty collection, depending on the matching objects and the **-intersect** option value.

For background on collections and querying of objects, see the **collections** man page.

COMMAND PHASING

The **remove_from_collection** command can be executed at any time.

EXAMPLES

The following example from NanoTime gets all input ports except "CLOCK".

```
nt_shell> set cPorts [remove_from_collection [all_inputs] CLOCK]
{"in1", "in2"}
```

SEE ALSO

add_to_collection(2)
collections(2)
query_objects(2)

remove_gated_clock_check

Removes gated-clock checks.

SYNTAX

```
status remove_gated_clock_check
   {-from from_object
      | -rise_from from_object
      | -fall_from from_object}
   {-to to_object
      | -rise_to to_object
      | -fall_to to_object}
   [-setup | -hold]
   [-of_objects timing_check_objects]
```

Data Types

```
   from_object             list
   to_object               list
   timing_check_objects    list
```

ARGUMENTS

-from *from_object*

Specifies a pin or port in the current design as the clock signal being gated. You must specify one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from *from_object*

Applies only to rising transitions at the clock pin. Similar to the **-from** option.

-fall_from *from_object*

Applies only to falling transitions at the clock pin. Similar to the **-from** option.

-to *to_object*

Specifies a pin or port in the current design as the clock-gating signal. You must specify one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to *to_object*

Applies only to rising signals at the clock-gating pin. Similar to the **-to** option.

-fall_to ***to_object***

Applies only to falling signals at the clock-gating pin. Similar to the **-to** option.

-setup

Removes the setup check originally specified with the **-setup** option.

-hold

Removes the hold check originally specified with the **-hold** option. If neither of the **-setup** or **-hold** options is specified, both checks are removed.

-of_objects ***timing_check_objects***

Removes gated clock timing checks from these timing check objects.

DESCRIPTION

This command removes gated clock timing checks, including both automatic topology-based checks and checks previously set with the **create_gated_clock_timing_check** command.

If you are removing checks set with the **create_gated_clock_timing_check** command, the **-from**, **-to**, **-setup** and **-hold** options must exactly match the ones used in the original **create_gated_clock_timing_check** command.

COMMAND PHASING

You can use the **remove_gated_clock_check** command at any time after a timing check is created and before executing a path tracing command (such as the **trace_paths** or **extract_model** command). If you use the **remove_gated_clock_check** command after the **trace_paths**, **extract_model** or **characterize_context** command, NanoTime transitions back to the "design-checked" state and discards all operations performed after the **check_design** command.

If you are removing an automatic topology-based check, the **remove_gated_clock_check** command would be executed after the **check_design** command because that is when the automatic timing checks are created.

If you are removing a manually created timing check, you can use the **remove_gated_clock_check** command at any time after executing the **create_gated_clock_timing_check** command that creates the timing check.

EXAMPLES

The following command removes both the setup and hold clock-gating checks set previously on ports CLK

and CLKGATE.

```
nt_shell> remove_gated_clock_check \
          -rise_from [get_ports CLK] \
          -to [get_ports CLKGATE]
```

SEE ALSO

create_gated_clock_timing_check(2)
set_gated_clock_timing_check_attributes(2)

remove_generated_clock

Removes a generated clock previously created by the **create_generated_clock** command.

SYNTAX

```
status remove_generated_clock
   [-all]
   [clock_list]
```

Data Types

```
   clock_list          list
```

ARGUMENTS

-all

Removes all clocks previously created by the **create_generated_clock** command.

clock_list

The list of generated clocks to be removed. It can be a list of one or more clock names or patterns that match clock names, or a list of collections containing clocks.

DESCRIPTION

Removes the generated clocks previously created by the **create_generated_clock** command. All attributes set on generated clocks (for example, by using the **false_paths**, **multicycle_paths**, or similar commands) are also removed.

To show a list all clocks currently defined in the design, use the **report_clock** command.

COMMAND PHASING

The **remove_generated_clock** command is typically executed in the "netlist-linked" state (in other

words, between the **link_design** and **check_topology** commands). It cannot be used before the **link_design** command. If you use the **remove_generated_clock** command after the **check_topology** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_topology** command.

If used in a lib_topologies definitions, the command should be in the pre_match_topology phase point.

EXAMPLES

The following command removes the generated clock GEN1 from the design.

```
nt_shell> remove_generated_clock GEN1
```

The following command removes all generated clocks from the design.

```
nt_shell> remove_generated_clock -all
```

SEE ALSO

```
create_generated_clock(2)
create_lib_topology(2)
report_clock(2)
topo_match_topology_reset_clock_and_topology(3)
```

remove_hspice_timing_model_paths

Removes all settings created by **set_hspice_timing_model_paths** commands.

SYNTAX

```
status remove_hspice_timing_model_paths
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **remove_hspice_timing_model_paths** command cancels the effects of all previous **set_hspice_timing_model_paths** commands.

SEE ALSO

```
extract_model(2)
set_hspice_timing_model_paths(2)
trace_paths(2)
```

remove_initial_condition_adjustment

Removes initial condition adjustments previously set with the **set_initial_condition_adjustment** command.

SYNTAX

```
status remove_initial_condition_adjustment
   objects
```

Data Types

```
objects        list
```

ARGUMENTS

objects

The list of nets from which to remove initial condition adjustments.

DESCRIPTION

The **remove_initial_condition_adjustment** command removes initial condition adjustments previously set with the **set_initial_condition_adjustment** command. You must specify a list of nets from which to the remove initial condition adjustments. After running the command, the initial conditions are set normally within the channel-connected blocks (CCBs) of the specified nets.

COMMAND PHASING

The **remove_initial_condition_adjustment** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_initial_condition_adjustment** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes the initial condition adjustment previously set within the CCB of a NAND gate. The net nand.out is the output of the NAND gate.

```
nt_shell> remove_initial_condition_adjustment \
          [get_nets nand.out]
```

SEE ALSO

set_initial_condition_adjustment(2)

remove_input_delay

Removes input delay information previously set on ports or pins with the **set_input_delay** command.

SYNTAX

```
status remove_input_delay
   [-clock clock_name]
   [-clock_fall]
   [-rise]
   [-fall]
   [-max]
   [-min]
   port_pin_list
```

Data Types

```
   clock_name            list
   port_pin_list         list
```

ARGUMENTS

-clock *clock_name*

Clock associated with the delay being removed. Use this option to remove input delay relative to one clock only.

-clock_fall

Removes the delay relative to falling edge of clock. If you specify *clock_name* without **-clock_fall**, the delay relative to rising edge of the clock is removed.

-rise

Removes input delay for rising transitions.

-fall

Removes input delay for falling transitions.

-max

Removes maximum input delay.

-min

Removes minimum input delay.

port_pin_list

Specifies a list of ports and pins from which to remove input delays. Each element in the list is either a collection of ports or pins, or a pattern which matches ports or pins on the current design.

DESCRIPTION

Removes input delay information from ports or pins in the current design previously set with the **set_input_delay** command. The default is to remove all input delay information in the *port_pin_list*.

To restrict the removed input delay values, use the **-clock**, **-clock_fall**, **-min**, **-max**, **-rise**, or **-fall** options.

To show input delays on ports, use the **report_port -input_delay** command.

COMMAND PHASING

The **remove_input_delay** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_input_delay** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example removes all input delay from ports IN*.

```
nt_shell> remove_input_delay [get_ports IN*]
```

The following example removes input delay relative to the rising edge of CLK from port DATA[5].

```
nt_shell> remove_input_delay -clock CLK { DATA[5] }
```

SEE ALSO

report_port(2)
set_input_delay(2)
set_output_delay(2)

```
remove_output_delay(2)
```

remove_input_noise

Removes a user-defined injected noise bump from a list of objects.

SYNTAX

```
status remove_input_noise
   [-above]
   [-below]
   [-high]
   [-low]
   objects
```

Data Types

```
   objects     list
```

ARGUMENTS

-above

Specifies a noise bump that goes above the quiet value of the victim net.

-below

Specifies a noise bump that goes below the quiet value of the victim net.

-high

Specifies a noise bump that occurs with the victim net held at Vdd.

-low

Specifies a noise bump that occurs with the victim net held at ground.

objects

Specifies the objects (ports or pins) from which the user-defined noise is to be removed.

DESCRIPTION

This command removes a user-defined injected noise bump from a list of ports or pins. The **-above, -below, -high**, and **-low** options can be used in combination to be more specific.

COMMAND PHASING

The **remove_input_noise** command can only be executed after the **trace_paths** command.

EXAMPLES

The following command removes an above high noise bump from pin Xxor7.MP1.g:

```
nt_shell> remove_input_noise -above -high Xxor7.MP1.g
```

SEE ALSO

```
report_noise(2)
report_fanout_noise(2)
remove_input_noise(2)
si_enable_noise_analysis(3)
```

remove_logic_constraint

Removes logic constraints previously set with the **set_logic_constraint** command.

SYNTAX

```
status remove_logic_constraint
   [-all]
   [-one_hot]
   [-one_off]
   [-at_most_one_hot]
   [-at_most_one_off]
   [-invert]
   [-equal]
   [-nand]
   [-nor]
   pins_or_nets
```

Data Types

```
   pins_or_nets         list
```

ARGUMENTS

-all

Removes all logic constraints from the specified objects. This is the default behavior of the command.

-one_hot

Removes only the "one hot" constraint from the specified objects.

-one_off

Removes only the "one off" constraint from the specified objects.

-at_most_one_hot

Removes only the "at most one hot" constraint from the specified objects.

-at_most_one_off

Removes only the "at most one off" constraint from the specified objects.

-invert

Removes only the "invert" constraint from the specified objects.

-equal

Removes only the "equal" constraint from the specified objects.

-nand

Removes only the "nand" constraint from the specified objects.

-nor

Removes only the "nor" constraint from the specified objects.

pins_or_nets

The list of related pins or nets from which to remove logic constraints.

DESCRIPTION

The **remove_logic_constraint** command removes logic constraints previously set on pins or nets with the **set_logic_constraint** command. You must specify the same list of pins or nets as the original **set_logic_constraint** command.

You can optionally specify the type of logic constraint (**-one_hot**, **-one_off**, and so on) to remove. Otherwise, all logic constraints are removed from the specified set of pins or nets.

Setting or removing logic constraints changes the design, making it necessary to run the **check_design** and **trace_paths** commands to get new analysis results.

COMMAND PHASING

The **remove_logic_constraint** command can only be executed in the "netlist-linked" state. If used after "netlist-linked", NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The following command removes the logic constraint previously set on signals ct0 through ct3.

```
nt_shell> remove_logic_constraint \
          [get_nets {ct0 ct1 ct2 ct3}]
```

SEE ALSO

report_logic_constraint(2)
set_logic_constraint(2)

remove_max_delay

Cancels the effects of the **set_max_delay** command by removing items from the current list of path exceptions.

SYNTAX

```
status remove_max_delay
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

```
   from_list          list
   rise_from_list     list
   fall_from_list     list
   to_list            list
   rise_to_list       list
   fall_to_list       list
   through_list       list
   rise_through_list  list
   fall_through_list  list
```

ARGUMENTS

-from *from_list*

Specifies a list of pins, ports, nets, or clocks considered path startpoints. If you specify a clock, all potential startpoints clocked by that clock are considered.

-rise_from *rise_from_list*

Same as the **-from** option, except that only rising-edge transitions from the specified objects are considered.

-fall_from *fall_from_list*

Similar to the **-rise_from** option, except that it is for falling-edge transitions.

-to ***to_list***

Specifies a list of pins, ports, nets, or clocks to be considered path endpoints. If you specify a clock, all potential endpoints clocked by that clock are considered.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_to ***fall_to_list***

Similar to the **-rise_to** option, except that it is for falling-edge transitions.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass in order to be considered.

-rise_through ***rise_through_list***

Same as the **-through** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_through ***fall_through_list***

Same as the **-through** option, except that only falling-edge transitions arriving at the specified objects are considered.

DESCRIPTION

The **remove_max_delay** command cancels the effects of the **set_max_delay** command by removing specified exceptions from the current set of exceptions listed by the **report_exceptions** command. The **remove_max_delay** and **set_max_delay** commands have the same **-from**, **-through**, and **-to** type options.

A general **remove_max_delay** command removes the effects of a matching general or more-specific **set_max_delay** command, as long as there is a matching object in the path specification of the original **set_max_delay** command. For example, suppose that you select the following paths:

```
nt_shell> set_max_delay -from selA -to BUS[60] 1.5
nt_shell> set_max_delay -from selB -to BUS[61] 2.0

nt_shell> report_exceptions

From               Through            To         Setup     Hold
-------------------------------------------------------------------
selA               *                  BUS[60]    max=1.5    --
selB               *                  BUS[61]    max=2.0    --
...
```

To remove the first item from the list of path restrictions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_max_delay -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

```
nt_shell> remove_max_delay -from selA
```

You could remove both sets of selected of paths by using the following command:

```
nt_shell> remove_max_delay -from sel*
```

This is because the **-from sel*** option matches the object specifications used in the original **set_max_delay** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_max_delay -through [get_nets n31]
nt_shell> remove_max_delay -rise_from selA
```

You cannot remove the subset of paths passing through net n31 because there is no matching object in the original **set_max_delay** command. You cannot remove just the rising-edge paths from selA because that is more specific than the **-from** option used in the original **set_max_delay** command.

In other words, you cannot remove a subset of paths specified by a single **set_max_delay** command.

COMMAND PHASING

The **remove_max_delay** command can be executed at any time after the **set_max_delay** command to which it is related.

EXAMPLES

The following example demonstrates how to selectively remove paths selected for path tracing.

```
nt_shell> set_max_delay 1.0 \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_max_delay 2.0 \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_max_delay 2.5 \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_exceptions

From               Through            To        Setup     Hold
-------------------------------------------  ---------------
selA               *                  BUS[60]   max=1.0    --
selB               *                  BUS[61]   max=2.0    --
clk1(r)            *                  clk2(f)   max=2.5    --

1
```

```
nt_shell> remove_max_delay \
          -from [get_clocks clk1]
1
nt_shell> report_exceptions

From               Through            To        Setup     Hold
-------------------------------------------  ----------------
selA               *                  BUS[60]   max=1.0    --
selB               *                  BUS[61]   max=2.0    --
```

The **remove_max_delay** command above removes the clock-to-clock path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** option.

```
nt_shell> remove_max_delay \
          -from [get_ports selA]

nt_shell> report_exceptions

From               Through            To
-----------------------------------------
selB               *                  BUS[61]
```

The **remove_max_delay** command above removes the path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** or **-to** specification.

SEE ALSO

check_design(2)
report_exceptions(2)
set_max_delay(2)
set_min_delay(2)
remove_min_delay(2)
trace_paths(2)

remove_measurement_threshold

Removes the delay and slew measurement thresholds previously set on specified nets with the **set_measurement_threshold** command.

SYNTAX

```
status remove_measurement_threshold
   objects
```

Data Types

objects	list

ARGUMENTS

objects

Specifies the nets from which to remove the delay and slew measurement parameters.

DESCRIPTION

The **set_measurement_threshold** command sets the delay and slew measurement thresholds for specified nets, overriding the parameter-set thresholds.

To cancel the effects of the **set_measurement_threshold** command, use the **remove_measurement_threshold** command.

COMMAND PHASING

The **remove_measurement_threshold** command must be executed after the **link_design** command but before a path tracing command such as the **trace_paths** or **extract_model** command.

EXAMPLES

The following command removes all net-specific measurement threshold parameters previously set on nets beginning with the characters "Sn".

```
nt_shell> remove_measurement_threshold \
          [get_nets Sn*]
```

SEE ALSO

set_measurement_threshold(2)

remove_memory

Removes a memory object in the current design.

SYNTAX

```
status remove_memory
   memory_list
```

Data Types

```
   memory_list     list
```

ARGUMENTS

memory_list

Removes memories for the specified list of memory objects. Provides the string, string pattern, list of strings or collection of memory objects.

DESCRIPTION

NanoTime for memories creates a memory object when manually specified by the **create_memory** command. Each memory object has a unique name. The **remove_memory** command resolves the memory list argument into a set of names and removes any memory objects with matching names from the current design. The memory_list can be a simple name string, a Tcl list of name strings, a collection of memory objects, or a name string pattern. All name string arguments must exactly match a memory object name string to be deleted.

COMMAND PHASING

The **remove_memory** command can be executed in any state after "netlist-linked" up to and including "paths-traced".

EXAMPLES

The following example removes all memory objects. The memory_list is a required argument; in this example it is a wildcard pattern for all memories.

```
nt_shell> remove_memory *
```

The following example takes a lookup pattern and removes any matches found.

```
nt_shell> remove_memory read_*
```

The following example takes a collection of all memories and erases them.

```
nt_shell> remove_memory [get_memory *]
```

SEE ALSO

```
create_memory(2)
get_memory(2)
report_memory(2)
```

remove_memory_bit_selected

Deselects bit cells for memory analysis.

SYNTAX

```
status remove_memory_bit_selected
   topology_list
```

Data Types

```
   topology_list     list
```

ARGUMENTS

topology_list

A list of ram topology objects to disable for memory analysis.

DESCRIPTION

NanoTime for memories allows custom selection of bit cells for analysis. To do this, use the **set_memory_analysis_type** command with the **-custom** option and then use the **set_memory_bit_selected** and **remove_memory_bit_selected** commands to create the set of selected bit cells.

The bit cells are designated by the ram topologies in the *topology_list*. A collection of ram topologies is created using the **get_topology** command. After execution, the ram topologies attribute *is_selected_bit* is set to true.

Use the **remove_memory_bit_selected** command to deselect a ram topology bit cell. After execution, the ram topology attribute *is_selected_bit* is set to false.

COMMAND PHASING

The **remove_memory_bit_selected** command can only be executed in the "topology-checked" state.

EXAMPLES

This command selects all the ram topologies in the collection for memory analysis.

```
nt_shell> remove_memory_bit_selected [get_topology -structure_type ram *]
```

SEE ALSO

```
get_topology(2)
set_memory_bit_selected(2)
remove_memory_bit_selected(2)
```

remove_merged_nets

Removes one or more nets from a group of nets previously merged with the **set_merged_nets** command.

SYNTAX

```
status remove_merged_nets
   net_list
```

Data Types

```
net_list          list
```

ARGUMENTS

net_list

One or more nets to be removed from a set of merged nets.

DESCRIPTION

This command removes a specified net (or a list of nets) from a set of nets previously merged with the **set_merged_nets** command.

COMMAND PHASING

The **remove_merged_nets** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_merged_nets** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes net c3 from the set of nets previously merged with the **set_merged_nets** command.

```
nt_shell> remove_merged_nets c3
```

SEE ALSO

set_merged_nets(2)
report_merged_nets(2)

remove_min_delay

Cancels the effects of the **set_min_delay** command by removing items from the current list of path exceptions.

SYNTAX

```
status remove_min_delay
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

```
   from_list           list
   rise_from_list      list
   fall_from_list      list
   to_list             list
   rise_to_list        list
   fall_to_list        list
   through_list        list
   rise_through_list   list
   fall_through_list   list
```

ARGUMENTS

-from *from_list*

Specifies a list of pins, ports, nets, or clocks considered path startpoints. If you specify a clock, all potential startpoints clocked by that clock are considered.

-rise_from *rise_from_list*

Same as the **-from** option, except that only rising-edge transitions from the specified objects are considered.

-fall_from *fall_from_list*

Similar to the **-rise_from** option, except that it is for falling-edge transitions.

-to ***to_list***

Specifies a list of pins, ports, nets, or clocks to be considered path endpoints, If you specify a clock, all potential endpoints clocked by that clock are considered.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_to ***fall_to_list***

Similar to the **-rise_to** option, except that it is for falling-edge transitions.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass to be considered.

-rise_through ***rise_through_list***

Same as the **-through** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_through ***fall_through_list***

Same as the **-through** option, except that only falling-edge transitions arriving at the specified objects are considered.

DESCRIPTION

The **remove_min_delay** command cancels the effects of the **set_min_delay** command by removing specified exceptions from the current sets of exceptions listed by the **report_exceptions** command. The **remove_min_delay** and **set_min_delay** commands have the same **-from**, **-through**, and **-to** type options.

A general **remove_min_delay** command removes the effects of a matching general or more-specific **set_min_delay** command, as long as there is a matching object in the path specification of the original **set_min_delay** command. For example, suppose that you select the following paths:

```
nt_shell> set_min_delay -from selA -to BUS[60] 1.5
nt_shell> set_min_delay -from selB -to BUS[61] 2.0

nt_shell> report_exceptions

From               Through            To          Setup     Hold
------------------------------------------------ ------------------
selA               *                  BUS[60]      --      min=1.5
selB               *                  BUS[61]      --      min=2.0
```

To remove the first item from the list of path restrictions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_min_delay -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

```
nt_shell> remove_min_delay -from selA
```

You could remove both sets of selected of paths by using the following command:

```
nt_shell> remove_min_delay -from sel*
```

This is because the **-from sel*** option matches the object specifications used in the original **set_min_delay** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_min_delay -through [get_nets n31]
nt_shell> remove_min_delay -rise_from selA
```

You cannot remove the subset of paths passing through net n31 because there is no matching object in the original **set_min_delay** command. You cannot remove just the rising-edge paths from selA because that is more specific than the **-from** option used in the original **set_min_delay** command.

In other words, you cannot remove a subset of paths specified by a single **set_min_delay** command.

COMMAND PHASING

The **remove_min_delay** command can be executed at any time after the **set_min_delay** command to which it is related.

EXAMPLES

The following example demonstrates how to selectively remove paths selected for path tracing.

```
nt_shell> set_min_delay 1.0 \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_min_delay 2.0 \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_min_delay 2.5 \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_exceptions

From               Through            To         Setup     Hold
-------------------------------------------  ---------------
selA               *                  BUS[60]     --      min=1.0
selB               *                  BUS[61]     --      min=2.0
clk1(r)            *                  clk2(f)     --      min=2.5

1
```

```
nt_shell> remove_min_delay \
          -from [get_clocks clk1]
1
nt_shell> report_exceptions
...
From               Through            To        Setup     Hold
-----------------------------------------------  ----------------
selA               *                  BUS[60]     --      min=1.0
selB               *                  BUS[61]     --      min=2.0
```

The **remove_min_delay** command above removes the clock-to-clock path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** option.

```
nt_shell> remove_min_delay \
          -from [get_ports selA]

nt_shell> report_exceptions

From               Through            To
-------------------------------------------
selB               *                  BUS[61]
```

The **remove_min_delay** command above removes the path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

check_design(2)
report_exceptions(2)
set_max_delay(2)
set_min_delay(2)
remove_max_delay(2)
trace_paths(2)

remove_min_pulse_width

Removes minimum pulse width constraints previously set with the **set_min_pulse_width** command.

SYNTAX

```
status remove_min_pulse_width
   -low | -high
   [-use_existing_timing_points]
   [object_list]
```

Data Types

```
   object_list       list
```

ARGUMENTS

-low

Removes the constraint on low pulses. Either the **-low** or **-high** option must be specified.

-high

Removes the constraint on high pulses. Either the **-low** or **-high** option must be specified.

-use_existing_timing_points

When this option is used the pins must already be timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used then the **check_design** command does not need to be run again after the **remove_min_pulse_width** command has been issued.

object_list

Specifies a list of clocks, cells, or pins in the current design from which to remove the constraint. If you do not use this option, the pulse width constraint is removed from all objects in the current design.

DESCRIPTION

The **remove_min_pulse_width** command removes minimum pulse width constraints previously set with the **set_min_pulse_width** command.

You can optionally specify a list of clocks, cells, or pins in the current design from which to remove the constraint. Otherwise, the command removes the pulse width constraint from all objects in the current design.

The **-low** or **-high** options allow you to specify which type of pulse_width constraints to remove.

COMMAND PHASING

The **remove_min_pulse_width** command can only be executed between "netlist-linked" and "topology-checked" states. If used after " topology-checked", NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed in the later phase(s).

However, if you use the **-use_existing_timing_points** option, you can use the **remove_min_pulse_width** command after the **check_design** command has been executed.

EXAMPLES

The following command removes the pulse width constraint from high pulses throughout the design.

```
nt_shell> remove_min_pulse_width -high
```

SEE ALSO

set_min_pulse_width(2)

remove_multicycle_path

Cancels the effects of a previous **set_multicycle_path** command.

SYNTAX

```
status remove_multicycle_path
   [-setup] [-hold] [-hold_cycle]
   [-rise] [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

from_list	list
rise_from_list	list
fall_from_list	list
to_list	list
rise_to_list	list
fall_to_list	list
through_list	list
rise_through_list	list
fall_through_list	list

ARGUMENTS

-setup

Removes a multicycle path timing exception previously set with a matching **-setup** option.

-hold

Removes a multicycle path timing exception previously set with a matching **-hold** or **-hold_cycle** option.

-hold_cycle

Removes a multicycle path timing exception previously set with a matching **-hold** or **-hold_cycle** option.

-rise

Removes a multicycle path timing exception previously set with a matching **-rise** option.

-fall

Removes a multicycle path timing exception previously set with a matching **-fall** option.

-from ***from_list***

Removes a multicycle path timing exception previously set with a matching **-from** option.

-rise_from ***rise_from_list***

Removes a multicycle path timing exception previously set with a matching **-rise_from** option.

-fall_from ***fall_from_list***

Removes a multicycle path timing exception previously set with a matching **-fall_from** option.

-to ***to_list***

Removes a multicycle path timing exception previously set with a matching **-to** option.

-rise_to ***rise_to_list***

Removes a multicycle path timing exception previously set with a matching **-rise_to** option.

-fall_to ***fall_to_list***

Removes a multicycle path timing exception previously set with a matching **-fall_to** option.

-through ***through_list***

Removes a multicycle path timing exception previously set with a matching **-through** option.

-rise_through ***rise_throughlist***

Removes a multicycle path timing exception previously set with a matching **-rise_through** option.

-fall_through ***fall_through_list***

Removes a multicycle path timing exception previously set with a matching **-fall_through** option.

DESCRIPTION

The **remove_multicycle_path** command cancels the effects of a previous **set_multicycle_path** command. It removes multicycle path definitions that can be listed with the **report_exceptions** command.

You must use at least one of the **-from**, **-through**, or **-to** type options to specify which multicycle path definitions to remove. A general **remove_multicycle_path** command removes the effects of a matching

general or more specific **set_multicycle_path** command, as long as there is a matching object in the path specification of the original **set_multicycle_path** command. For example, suppose that you define the following multicycle paths:

```
nt_shell> set_multicycle_path 2 \
          -from selA -to BUS[60]
nt_shell> set_multicycle_path 2 \
          -from selB -to BUS[61]
nt_shell> report_exceptions

From    Through   To          Setup       Hold
-------------------------------------------------
selA    *         BUS[60]     cycle=2     cycle=2
selB    *         BUS[61]     cycle=2     cycle=2
```

To remove the first multicycle path definition from the list of exceptions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_multicycle_path \
          -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

nt_shell> **remove_multicycle_path -from selA**

You could remove both sets of selected of paths by using the following command:

nt_shell> **remove_multicycle_path -from sel***

This is because the **-from sel*** option matches the object specifications used in the original **set_multicycle_path** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_multicycle_path \
          -through [get_nets n31]
nt_shell> remove_multicycle_path \
          -rise_from selA
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_multicycle_path** command. You cannot remove just the rising-edge path exceptions from selA because that is more specific than the **-from** option used in the original **set_multicycle_path** command.

In other words, you cannot remove a subset of multicycle path definitions specified by a single **set_multicycle_path** command. You can only remove whole items listed in the **report_exceptions** report.

COMMAND PHASING

The **remove_multicycle_path** command can be executed any time after the **set_multicycle_path** command to which it relates.

EXAMPLES

The following example demonstrates how to selectively remove multicycle path definitions.

```
nt_shell> set_multicycle_path 2 \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_multicycle_path 2 \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_multicycle_path 2 \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_exceptions

From      Through   To        Setup       Hold
-------------------------------------------------
selA      *         BUS[60]   cycle=2     cycle=2
selB      *         BUS[61]   cycle=2     cycle=2
clk1(r)   *         clk2(f)   cycle=2     cycle=2

1
nt_shell> remove_multicycle_path \
          -from [get_clocks clk1]
1
nt_shell> report_exceptions

From      Through   To         Setup       Hold
-------------------------------------------------
selA      *         BUS[60]    cycle=2     cycle=2
selB      *         BUS[61]    cycle=2     cycle=2
```

The **remove_multicycle_path** command above removes the clock-to-clock multicycle path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** option.

```
nt_shell> remove_multicycle_path \
          -from [get_ports selA]

nt_shell> report_exceptions

From      Through   To         Setup       Hold
-------------------------------------------------
selB      *         BUS[61]    cycle=2     cycle=2
```

The **remove_multicycle_path** command above removes the multicycle path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

```
report_exceptions(2)
set_multicycle_path(2)
```

remove_net_group

Removes one or more net groups previously created by the **create_net_group** command.

SYNTAX

```
string remove_net_group
   net_group_list
```

Data Types

net_group_list	list

ARGUMENTS

net_group_list

The list of net groups to be removed.

DESCRIPTION

The **remove_net_group** command removes one or more net groups previously created by the **create_net_group** command. The command only affects the grouping of the nets, not the nets themselves.

COMMAND PHASING

The **remove_net_group** command can be executed any time after the **create_net_group** command to which it relates.

EXAMPLES

The following command removes the net group called SBUS.

```
nt_shell> remove_net_group {SBUS}
```

SEE ALSO

create_net_group(2)
list_net_groups(2)

remove_netlist

Removes netlists from the netlist registry.

SYNTAX

```
status remove_netlist
   file_names
```

Data Types

```
   file_names          list
```

ARGUMENTS

file_names

Specifies the names of the netlist files to remove from the netlist registry.

DESCRIPTION

The **remove_netlist** command removes netlists that were previously specified with the **register_netlist** command from the netlist registry. All matching names are removed. You can use wildcard characters to remove multiple netlist entries.

You must specify the full path name of the netlist files to be removed. To view a list of registered netlist files, including the full path names, use the **list_netlists** command.

COMMAND PHASING

The **remove_netlist** command is not phase-restricted.

EXAMPLES

The following example lists the registered netlist files, then removes the netlist file "alu_lat.sp".

```
nt_shell> list_netlists
Netlist Registry:

     Format     File Name
---- ---------- ----------------------------------
 *L  spice      /remote/mydir/designs/alu_lat.sp
 *L  spice      /remote/mydir/designs/dtech.sp
1
nt_shell> remove_netlist /remote/mydir/designs/alu*
1
```

SEE ALSO

list_netlists(2)
register_netlist(2)

remove_no_same_phase_path

Cancels the effects of a previous **set_no_same_phase_path** command.

SYNTAX

```
status remove_no_same_phase_path
   [-setup] [-hold]
   [-rise] [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

```
   from_list               list
   rise_from_list          list
   fall_from_list          list
   to_list                 list
   rise_to_list            list
   fall_to_list            list
   through_list            list
   rise_through_list       list
   fall_through_list       list
```

ARGUMENTS

-setup

Removes a no-same-phase path timing exception previously set with a matching **-setup** option.

-hold

Removes a no-same-phase path timing exception previously set with a matching **-hold** option.

-rise

Removes a no-same-phase path timing exception previously set with a matching **-rise** option.

-fall

Removes a no-same-phase path timing exception previously set with a matching **-fall** option.

-from *list*

Removes a no-same-phase path timing exception previously set with a matching **-from** option.

-rise_from *list*

Removes a no-same-phase path timing exception previously set with a matching **-rise_from** option.

-fall_from *list*

Removes a no-same-phase path timing exception previously set with a matching **-fall_from** option.

-to *list*

Removes a no-same-phase path timing exception previously set with a matching **-to** option.

-rise_to *list*

Removes a no-same-phase path timing exception previously set with a matching **-rise_to** option.

-fall_to *list*

Removes a no-same-phase path timing exception previously set with a matching **-fall_to** option.

-through *list*

Removes a no-same-phase path timing exception previously set with a matching **-through** option.

-rise_through *list*

Removes a no-same-phase path timing exception previously set with a matching **-rise_through** option.

-fall_through *list*

Removes a no-same-phase path timing exception previously set with a matching **-fall_through** option.

DESCRIPTION

The **remove_no_same_phase_path** command cancels the effects of a previous **set_no_same_phase_path** command. It removes no-same-phase path definitions that can be listed with the **report_exceptions** command.

You must use at least one of the **-from**, **-through**, or **-to** type options to specify which no-same-phase path definitions to remove. A general **remove_no_same_phase_path** command removes the effects of a matching general or more specific **set_no_same_phase_path** command, as long as there is a matching object in the path specification of the original **set_no_same_phase_path** command. For example, suppose that you define the following no-same-phase paths:

```
nt_shell> set_no_same_phase_path \
          -from selA -to BUS[60]
```

```
nt_shell> set_no_same_phase_path \
          -from selB -to BUS[61]
nt_shell> report_exceptions
...
From    Through   To         Setup          Hold
-------------------------------------------------
selA    *         BUS[60]    no_same_phase  ...
selB    *         BUS[61]    no_same_phase  ...
```

To remove the first no-same-phase path definition from the list of exceptions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_no_same_phase_path \
          -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

nt_shell> **remove_no_same_phase_path -from selA**

You could remove both sets of selected of paths by using the following command:

nt_shell> **remove_no_same_phase_path -from sel***

This is because the **-from sel*** option matches the object specifications used in the original **set_no_same_phase_path** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_no_same_phase_path \
          -through [get_nets n31]
nt_shell> remove_no_same_phase_path \
          -rise_from selA
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_no_same_phase_path** command. You cannot remove just the rising-edge path exceptions from selA because that is more specific than the **-from** option used in the original **set_no_same_phase_path** command.

In other words, you cannot remove a subset of no-same-phase path definitions specified by a single **set_no_same_phase_path** command. You can only remove whole items listed in the **report_exceptions** report.

COMMAND PHASING

The **remove_no_same_phase_path** command can be executed any time after the **set_no_same_phase_path** command to which it relates.

EXAMPLES

The following example demonstrates how to selectively remove no-same-phase path definitions.

```
nt_shell> set_no_same_phase_path \
          -from [get_ports selA] \
```

```
          -to [get_ports BUS[60]]
1
nt_shell> set_no_same_phase_path \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_no_same_phase_path \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_exceptions

From      Through   To        Setup          Hold
-----------------------------------------------------
selA      *         BUS[60]   no_same_phase  ...
selB      *         BUS[61]   no_same_phase  ...
clk1(r)   *         clk2(f)   no_same_phase  ...

1
nt_shell> remove_no_same_phase_path \
          -from [get_clocks clk1]
1
nt_shell> report_exceptions

From      Through   To         Setup          Hold
-------------------------------------------------------
selA      *         BUS[60]    no_same_phase  ...
selB      *         BUS[61]    no_same_phase  ...
```

The **remove_no_same_phase_path** command above removes the clock-to-clock no-same-phase path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** specification.

```
nt_shell> remove_no_same_phase_path \
          -from [get_ports selA]

nt_shell> report_exceptions
...
From      Through   To         Setup          Hold
-------------------------------------------------------
selB      *         BUS[61]    no_same_phase  ...
```

The **remove_no_same_phase_path** command above removes the no_same_phase path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

report_exceptions(2)
set_no_same_phase_path(2)

remove_no_timing_check_path

Cancels the effects of a previous **set_no_timing_check_path** command.

SYNTAX

```
status remove_no_timing_check_path
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   -to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list
```

Data Types

```
   from_list               list
   rise_from_list          list
   fall_from_list          list
   through_list            list
   rise_through_list       list
   fall_through_list       list
   to_list                 list
   rise_to_list            list
   fall_to_list            list
```

ARGUMENTS

-setup

Removes a no-timing-check path timing exception previously set with a matching **-setup** option.

-hold

Removes a no-timing-check path timing exception previously set with a matching **-hold** option.

-rise

Removes a no-timing-check path timing exception previously set with a matching **-rise** option.

-fall

Removes a no-timing-check path timing exception previously set with a matching **-fall** option.

-from ***list***

Removes a no-timing-check path timing exception previously set with a matching **-from** option.

-rise_from ***list***

Removes a no-timing-check path timing exception previously set with a matching **-rise_from** option.

-fall_from ***list***

Removes a no-timing-check path timing exception previously set with a matching **-fall_from** option.

-through ***list***

Removes a no-timing-check path timing exception previously set with a matching **-through** option.

-rise_through ***list***

Removes a no-timing-check path timing exception previously set with a matching **-rise_through** option.

-fall_through ***list***

Removes a no-timing-check path timing exception previously set with a matching **-fall_through** option.

-to ***list***

Removes a no-timing-check path timing exception previously set with a matching **-to** option.

-rise_to ***list***

Removes a no-timing-check path timing exception previously set with a matching **-rise_to** option.

-fall_to ***list***

Removes a no-timing-check path timing exception previously set with a matching **-fall_to** option.

DESCRIPTION

The **remove_no_timing_check_path** command cancels the effects of a previous **set_no_timing_check_path** command. It removes no-timing-check path definitions that can be listed with the **report_exceptions** command.

You must use at least one of the **-from**, **-through**, or **-to** type options to specify which no-timing-check path definitions to remove. A general **remove_no_timing_check_path** command removes the effects of a matching general or more specific **set_no_timing_check_path** command, as long as there is a matching object in the path specification of the original **set_no_timing_check_path** command. For

example, suppose that you define the following no-timing-check paths:

```
nt_shell> set_no_timing_check_path \
          -from selA -to BUS[60]
nt_shell> set_no_timing_check_path \
          -from selB -to BUS[61]
nt_shell> report_exceptions

From    Through   To        Setup     Hold       ...
-------------------------------------------------
selA    *         BUS[60]   no_check  no_check   ...
selB    *         BUS[61]   no_check  no_check   ...
```

To remove the first no-timing-check path definition from the list of exceptions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_no_timing_check_path \
          -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

```
nt_shell> remove_no_timing_check_path -from selA
```

You could remove both sets of selected of paths by using the following command:

```
nt_shell> remove_no_timing_check_path -from sel*
```

This is because the **-from sel*** option matches the object specifications used in the original **set_no_timing_check_path** commands.

However, the following commands would not have any effect:

```
nt_shell> remove_no_timing_check_path \
          -through [get_nets n31]
nt_shell> remove_no_timing_check_path \
          -rise_from selA
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_no_timing_check_path** command. You cannot remove just the rising-edge path exceptions from selA because that is more specific than the **-from** option used in the original **set_no_timing_check_path** command.

In other words, you cannot remove a subset of no-timing-check path definitions specified by a single **set_no_timing_check_path** command. You can only remove whole items listed in the **report_exceptions** report.

COMMAND PHASING

The **remove_no_timing_check_path** command can be executed any time after the **set_no_timing_check_path** command to which it relates.

EXAMPLES

The following example demonstrates how to selectively remove no-timing-check path definitions.

```
nt_shell> set_no_timing_check_path \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_no_timing_check_path \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_no_timing_check_path \
          -rise_from [get_clocks clk2] \
          -fall_to [get_clocks clk3]
1
nt_shell> report_exceptions

From      Through   To       Setup     Hold       ...
-----------------------------------------------------
selA      *         BUS[60]  no_check  no_check   ...
selB      *         BUS[61]  no_check  no_check   ...
clk2(r)   *         clk3(f)  no_check  no_check   ...

1
nt_shell> remove_no_timing_check_path \
          -from [get_clocks clk2]
1
nt_shell> report_exceptions

From      Through   To       Setup     Hold       ...
-----------------------------------------------------
selA      *         BUS[60]  no_check  no_check   ...
selB      *         BUS[61]  no_check  no_check   ...
```

The **remove_no_timing_check_path** command above removes the clock-to-clock no-check path specification because there is a matching object (clock clk1) and the **-from** option is more general than the **-rise_from** option.

```
nt_shell> remove_no_timing_check_path \
          -from [get_ports selA]

nt_shell> report_exceptions

From      Through   To       Setup     Hold       ...
-----------------------------------------------------
selB      *         BUS[61]  no_check  no_check   ...
```

The **remove_no_timing_check_path** command above removes the no-check path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

report_exceptions(2)
set_no_timing_check_path(2)

remove_non_transparent

Removes nontransparent checking behavior previously set with the **set_non_transparent** command.

SYNTAX

```
status remove_non_transparent
   object_list
```

Data Types

object_list	list

ARGUMENTS

object_list

Specifies the topologies and clocks from which to remove nontransparent checking behavior.

DESCRIPTION

This command removes nontransparent checking behavior previously with the **set_non_transparent** command. In the command, you must specify topologies or clocks in the design from which to remove nontransparent checking behavior.

COMMAND PHASING

The **remove_non_transparent** command must be executed after the **link_design** command but before a path tracing command such as the **trace_paths** or **extract_model** command.

EXAMPLES

The following command removes nontransparent checking from the latch that has its latch node at net n1.

```
nt_shell> remove_non_transparent [get_topology \
          -of_object n1 -structure_type latch]
```

SEE ALSO

set_non_transparent(2)

remove_output_delay

Removes output delay information previously set on ports or pins with the **set_output_delay** command.

SYNTAX

```
status remove_output_delay
   [-clock clock_name]
   [-clock_fall]
   [-level_sensitive]
   [-rise]
   [-fall]
   [-max]
   [-min]
   port_pin_list
```

Data Types

```
   clock_name          string
   port_pin_list       list
```

ARGUMENTS

-clock *clock_name*

Clock associated with the delay being removed. Use this option to remove output delay relative to one clock only.

-clock_fall

Removes the delay relative to falling edge of clock. If you specify *clock_name* without **-clock_fall**, the delay relative to rising edge of the clock is removed.

-level_sensitive

Specifies that the source of the delay is a level-sensitive latch.

-rise

Removes output delay for rising transitions.

-fall

Removes output delay for falling transitions.

-max

Removes maximum output delay.

-min

Removes minimum output delay.

port_pin_list

Specifies a list of ports and pins. Each element in the list is either a collection of ports or pins, or a pattern which matches ports or pins on the current design.

DESCRIPTION

Removes output delay values for objects in the current design previously set with the **set_output_delay** command. By default, all output delays of objects in *port_pin_list* are removed.

To restrict the removed output delay values, use the **-clock**, **-clock_fall**, **-min**, **-max**, **-rise**, or **-fall** option.

To show output delays associated with ports, use the **report_port -output_delay** command.

COMMAND PHASING

The **remove_output_delay** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_output_delay** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example removes all output delay values from all output ports in the current design.

```
nt_shell> remove_output_delay [all_outputs]
```

The following example removes maximum output delay values for rising edges from OUT1, OUT3, and BIDIR12.

```
nt_shell> remove_output_delay -max -rise \
          { OUT1 OUT3 BIDIR12 }
```

The following example removes all output delays for port OUT1 relative to the falling edge of CLK2.

```
nt_shell> remove_output_delay -clock CLK2 \
```

```
                -clock_fall OUT1
```

The following example removes all output delays not relative to any clock for port OUT2.

```
nt_shell> remove_output_delay -clock {""} OUT2
```

SEE ALSO

```
all_outputs(2)
collections(2)
report_port(2)
set_output_delay(2)
set_input_delay(2)
```

remove_pattern

Removes a pattern netlist previously read in with the **read_pattern** command.

SYNTAX

```
status remove_pattern
   [-all]
   name
```

Data Types

```
   name        string
```

ARGUMENTS

-all

Removes all pattern netlists in the pattern registry.

name

Removes only the named patterns.

DESCRIPTION

This command removes a named pattern or all patterns from the pattern registry.

COMMAND PHASING

The **remove_pattern** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example removes all patterns from the pattern registry.

```
nt_shell> remove_pattern -all
```

The following example removes pattern XOR from the pattern registry.

```
nt_shell> remove_pattern XOR
```

SEE ALSO

```
list_patterns(2)
read_pattern(2)
```

remove_pbsa_override

Removes the effects of a previous **set_pbsa_override** command.

SYNTAX

```
string remove_pbsa_override
   [-from from_list]
   [-to to_list]
   [-through through_list]
```

Data Types

```
   from_list     list
   to_list       list
   through_list  list
```

ARGUMENTS

-from *from_list*

Removes a path based slack adjustment exception previously set with a matching **-from** option on a **set_pbsa_override** command.

-to *to_list*

Removes a path based slack adjustment exception previously set with a matching **-to** option on a **set_pbsa_override** command.

-through *through_list*

Removes a path based slack adjustment exception previously set with a matching **-through** option on a **set_pbsa_override** command.

DESCRIPTION

The **remove_pbsa_override** command cancels the effects of a previous **set_pbsa_override**

command.

You must use at least one of the **-from**, **-through**, or **-to** options to specify which path overrides to remove. A general **remove_pbsa_override** command removes the effects of a matching general or more specific **set_pbsa_override** command, if there is a matching object in the path specification of the original **set_pbsa_override** command.

EXAMPLES

Suppose that you define the following path based slack adjustment overrides:

```
nt_shell> set_pbsa_override -from selA -to BUS[60]
nt_shell> set_pbsa_override -from selB -to BUS[61]
nt_shell> report_exceptions
...

From     Through   To         Setup   Hold     Matches
-------------------------------------------------------
selA     *         BUS[60]    FALSE   FALSE    --
selB     *         BUS[61]    FALSE   FALSE    --
```

To remove the first path definition from the list of exceptions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_pbsa_override -from selA -to BUS[60]
```

The following command would have the same effect because it specifies the paths in a less specific manner:

```
nt_shell> remove_pbsa_override -from selA
```

You could remove both sets of paths by using the following command:

```
nt_shell> remove_pbsa_override -from sel*
```

This is because "-from sel*" matches the object specifications used in the original **set_pbsa_override** commands.

However, the following command would not have any effect:

```
nt_shell> remove_pbsa_override -through [get_nets n31]
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_pbsa_override** command. In other words, you cannot remove a subset of path definitions specified by a single **set_pbsa_override** command.

SEE ALSO

set_pbsa_override(2)

remove_phasing

Removes same-cycle or next-cycle checking previously set on objects with the **set_phasing** command.

SYNTAX

```
status remove_phasing
   [-setup]
   [-hold]
   object_list
```

Data Types

```
   object_list          list
```

ARGUMENTS

-setup

Removes the previous phasing setting for setup checks only. In the absence of the **-setup** and **-hold** options, both setup and hold settings are removed.

-hold

Removes the previous phasing setting for hold checks only.

object_list

Specifies the objects from which to remove the previous same-cycle or next-cycle phasing setting.

DESCRIPTION

The **set_phasing** command selects same-cycle or next-cycle checking for specified instances of objects in the design where sequential timing checks are performed. The **remove_phasing** command reverses the effects of any previous **set_phasing** command for specified instances of objects in the design. Removing these object-based settings allows the global default setting to take effect for those objects. The

global default setting is controlled by the **timing_default_phasing** variable.

For more information about same-cycle and next-cycle checking, see the man pages for the set_phasing command and the **timing_default_phasing** variable.

COMMAND PHASING

The **remove_phasing** command can be executed any time after the **set_phasing** command to which it relates.

EXAMPLES

The following command removes any previous same-cycle or next-cycle setting on the output called out1.

```
nt_shell> remove_phasing [get_ports out1]
1
```

The following command removes any previous same-cycle or next-cycle setting on the topology called latch1.

```
nt_shell> remove_phasing [get_topology \
          -all -structure_type latch \
          -filter "structure_name == latch1"]
```

The following command removes any previous same-cycle or next-cycle settings on all latches in the design.

```
nt_shell> remove_phasing [get_topology \
          -all -structure_type latch]
```

SEE ALSO

```
get_topology(2)
remove_no_same_phase_path(2)
remove_same_phase_path(2)
set_no_same_phase_path(2)
set_phasing(2)
set_same_phase_path(2)
timing_default_phasing(3)
```

remove_pin_variation

Removes user defined variation values previously set with the **set_pin_variation** command.

SYNTAX

```
status remove_pin_variation
   objects
```

Data Types

```
   objects        list
```

ARGUMENTS

objects

The list of nets or transistor gate pins affected by the command. This list defines a set of delay calculation trigger pins or nets. The variation values from these trigger pins or nets are no longer overridden.

DESCRIPTION

The **remove_pin_variation** command removes variation values previously set with the **set_pin_variation** command. You must specify a list of nets or transistor gates from which to remove the variation values. After running the command, the variation values are calculated without any user modification at the specified nets or transistor gates.

COMMAND PHASING

The **remove_pin_variation** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **remove_pin_variation** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes all variation values previously set on pin X1.MP1.G .

```
nt_shell> remove_pin_variation [get_nets X1.MP1.G]
```

SEE ALSO

set_pin_variation(2)
report_pin_variation(2)

remove_precharge_input_net

Removes the precharge-only specification previously set on a net with the **set_precharge_input_net** command.

SYNTAX

```
status remove_precharge_input_net
   net_list
```

Data Types

net_list	list

ARGUMENTS

net_list

Specifies the nets from which to remove the precharge-only designation.

DESCRIPTION

This command removes the precharge-only specification previously set on one or more nets with the **set_precharge_input_net** command.

COMMAND PHASING

The **remove_precharge_input_net** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_precharge_input_net** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes the precharge-only designation from net n1.

```
nt_shell> remove_precharge_input_net [get_nets n1]
```

SEE ALSO

set_precharge_input_net(2)

remove_propagated_clock

Removes propagated clocking behavior and invokes ideal clocking for specified objects.

SYNTAX

```
status remove_propagated_clock
   object_list
```

Data Types

```
   object_list             list
```

ARGUMENTS

object_list

A list of clocks, ports, or pins that are to use ideal (rather than propagated) clocking. If you specify a clock, the whole clock network becomes ideal. If you specify a port or pin, ideal clocking is used for all objects in the transitive fanout of the specified port or pin.

DESCRIPTION

The **remove_propagated_clock** command causes specified clocks, ports, or pins to use ideal (rather than propagated) clocking. Propagated clocking is the default behavior in NanoTime.

For each sequential element in the design, NanoTime uses either propagated or ideal clocking. 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.

To change a clock, port, or pin to use ideal clocking, use the **remove_propagated_clock** command or use the **set_clock_latency** command to specify an ideal latency value. To restore propagated clocking behavior, use the **set_propagated_clock** command.

To see propagated clock attributes on clocks, use the **report_clock** command. For information about

propagated clock attributes on clocks, see the man page for the **report_clock** command.

For more information about propagated versus ideal clocking, see the man page for the **set_clock_latency** command.

COMMAND PHASING

The **remove_propagated_clock** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** and **check_design** commands). It cannot be used before the **link_design** command. If you use the **remove_propagated_clock** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes propagated clock behavior and invokes ideal clocking for all clocks in the design.

nt_shell> **remove_propagated_clock [all_clocks]**

SEE ALSO

report_clock(2)
set_clock_latency(2)
set_propagated_clock(2)

remove_requires_clock

Removes the is_requires_clock attribute.

SYNTAX

```
status remove_requires_clock
   object_list
```

Data Types

```
   object_list          collection
```

ARGUMENTS

object_list

The list or collection of ports, nets and pins to be modified.

DESCRIPTION

This command removes the **is_requires_clock** attribute from the specified objects. The **is_requires_clock** attribute is an attribute placed on objects by the **set_requires_clock** command to help troubleshoot clock networks.

COMMAND PHASING

The **remove_requires_clock** command is typically executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command).

You cannot use the command before the **link_design** command. If you use the **remove_requires_clock** command after the **check_topology** command, the NanoTime tool transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following example removes the **is_requires_clock** attribute from net latch_out.

```
nt_shell> remove_requires_clock "latch_out"
```

SEE ALSO

report_clock_network(2)
set_requires_clock(2)

remove_same_phase_path

Cancels the effects of a previous **set_same_phase_path** command.

SYNTAX

```
status remove_same_phase_path
   [-setup] [-hold]
   [-rise] [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
```

Data Types

```
   from_list           list
   rise_from_list      list
   fall_from_list      list
   to_list             list
   rise_to_list        list
   fall_to_list        list
   through_list        list
   rise_through_list   list
   fall_through_list   list
```

ARGUMENTS

-setup

Removes a same-phase path timing exception previously set with a matching **-setup** option.

-hold

Removes a same-phase path timing exception previously set with a matching **-hold** option.

-rise

Removes a same-phase path timing exception previously set with a matching **-rise** option.

-fall

Removes a same-phase path timing exception previously set with a matching **-fall** option.

-from *list*

Removes a same-phase path timing exception previously set with a matching **-from** option.

-rise_from *list*

Removes a same-phase path timing exception previously set with a matching **-rise_from** option.

-fall_from *list*

Removes a same-phase path timing exception previously set with a matching **-fall_from** option.

-to *list*

Removes a same-phase path timing exception previously set with a matching **-to** option.

-rise_to *list*

Removes a same-phase path timing exception previously set with a matching **-rise_to** option.

-fall_to *list*

Removes a same-phase path timing exception previously set with a matching **-fall_to** option.

-through *list*

Removes a same-phase path timing exception previously set with a matching **-through** option.

-rise_through *list*

Removes a same-phase path timing exception previously set with a matching **-rise_through** option.

-fall_through *list*

Removes a same-phase path timing exception previously set with a matching **-fall_through** option.

DESCRIPTION

The **remove_same_phase_path** command cancels the effects of a previous **set_same_phase_path** command. It removes same-phase path definitions that can be listed with the **report_exceptions** command.

You must use at least one of the **-from**, **-through**, or **-to** type options to specify which same-phase path definitions to remove. A general **remove_same_phase_path** command removes the effects of a matching general or more specific **set_same_phase_path** command, as long as there is a matching object in the path specification of the original **set_same_phase_path** command. For example, suppose that you define the following same-phase paths:

```
nt_shell> set_same_phase_path \
          -from selA -to BUS[60]
nt_shell> set_same_phase_path \
```

```
          -from selB -to BUS[61]
nt_shell> report_exceptions

From    Through   To         Setup       Hold
-------------------------------------------------
selA    *         BUS[60]    same_phase  same_phase
selB    *         BUS[61]    same_phase  same_phase
```

To remove the first same-phase path definition from the list of exceptions (from selA to BUS[60]), you could use the following command:

```
nt_shell> remove_same_phase_path \
          -from selA -to BUS[60]
```

The following command would have the same effect in this example because it specifies the paths in a less specific manner:

nt_shell> **remove_same_phase_path -from selA**

You could remove both sets of selected of paths by using the following command:

nt_shell> **remove_same_phase_path -from sel***

This is because the **-from sel*** option matches the object specifications used in the original **set_same_phase_path** command.

However, the following commands would not have any effect:

```
nt_shell> remove_same_phase_path \
          -through [get_nets n31]
nt_shell> remove_same_phase_path \
          -rise_from selA
```

You cannot remove the subset of path exceptions passing through net n31 because there is no matching object in the original **set_same_phase_path** command. You cannot remove just the rising-edge path exceptions from selA, because that is more specific than the **-from** option used in the original **set_same_phase_path** command.

In other words, you cannot remove a subset of same-phase path definitions specified by a single **set_same_phase_path** command. You can only remove whole items listed in the **report_exceptions** report.

COMMAND PHASING

The **remove_same_phase_path** command can be executed any time after the **set_same_phase_path** command to which it relates.

EXAMPLES

The following example demonstrates how to selectively remove same-phase path definitions.

```
nt_shell> set_same_phase_path \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
```

```
1
nt_shell> set_same_phase_path \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> set_same_phase_path \
          -rise_from [get_clocks clk1] \
          -fall_to [get_clocks clk2]
1
nt_shell> report_exceptions

From      Through   To         Setup       Hold
-----------------------------------------------------
selA      *         BUS[60]   same_phase  same_phase
selB      *         BUS[61]   same_phase  same_phase
clk1(r)   *         clk2(f)   same_phase  same_phase

1
nt_shell> remove_same_phase_path \
          -from [get_clocks clk1]
1
nt_shell> report_exceptions

From      Through   To          Setup       Hold
-------------------------------------------------------
selA      *         BUS[60]    same_phase  same_phase
selB      *         BUS[61]    same_phase  same_phase
```

The **remove_same_phase_path** command above removes the clock-to-clock same-phase path specification because there is a matching object (clock clk1) and the **-from** specification is more general than the **-rise_from** specification.

```
nt_shell> remove_same_phase_path \
          -from [get_ports selA]

nt_shell> report_exceptions
...
From      Through   To          Setup       Hold
-------------------------------------------------------
selB      *         BUS[61]    same_phase  same_phase
```

The **remove_same_phase_path** command above removes the same-phase path specification because there is a matching object (in the **-from selA** option) and the **-from** specification is more general than the original **-from** and **-to** specifications.

SEE ALSO

report_exceptions(2)
set_same_phase_path(2)

remove_search_enabled

Disables the searching of specified lib_topology objects.

SYNTAX

```
status remove_search_enabled
   [lib_topology_list]
```

Data Types

```
   lib_topology_list          list
```

ARGUMENTS

lib_topology_list

A collection of lib_topology objects created with the **get_lib_topology** command. NanoTime disables searching of the specified lib_topology objects.

DESCRIPTION

This command disables specified lib_topology objects for searching. The effect of the command is to set the **search_enabled** attribute of the specified lib_topology objects to **false**. When you later search for topologies with the **match_topology** or **check_topology** command, NanoTime searches only for the lib_topology objects that have their corresponding **search_enabled** attributes set to **true**.

In the **set_search_enabled** command, use the **get_lib_topology** command to create the collection of lib_topology objects to be disabled. Specify the lib_topology objects in the following format:

library_name.pattern

where *library_name* is the name of the topology library, the period is the hierarchy separator character (which can be changed by setting the **hierarchy_separator** variable) and *pattern* is the lib_topology name or a wildcard pattern that matches one or more lib_topology names.

The **set_search_enabled** command has the opposite effect. It sets the **search_enabled** attribute to **true** for specified topology objects.

COMMAND PHASING

The **remove_search_enabled** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command disables searching of the mux3 lib_topology in the user2 topology library.

```
nt_shell> remove_search_enabled \
          [get_lib_topology user2.mux3]
```

The following command creates a collection of global lib_topology objects for which searching is currently disabled:

```
nt_shell> get_lib_topology \
          -filter "search_enabled==false" global.*
```

SEE ALSO

check_topology(2)
get_lib_topology(2)
list_lib_topology(2)
match_topology(2)
read_topology_library(2)
set_search_enabled(2)

remove_si_delay_analysis

Removes the reselection parameters previously set on nets with the **set_si_delay_analysis** command.

SYNTAX

```
status remove_si_delay_analysis
   [-reselect rnets]
   [-ignore_arrival inets]
   [-exclude]
   [-victims vnets]
   [-aggressors anets]
   [-min]
   [-max]
   [-all]
```

Data Types

```
   rnets         list
   inets         list
   vnets         list
   anets         list
```

ARGUMENTS

-reselect *rnets*

Removes the effects of the **set_si_delay_analysis** command with the **-reselect** option.

-ignore_arrival *inets*

Removes the effects of the **set_si_delay_analysis** command with the **-ignore_arrival** option.

-exclude

Removes the effects of the **set_si_delay_analysis** command with the **-exclude** option.

-victims *vnets*

With the **-exclude** option, specifies a list of nets from which to remove the victim exclusion setting.

-aggressors *anets*

With the **-exclude** option, specifies a list of nets from which to remove the aggressor exclusion setting.

-min

With the **-exclude** option, removes the exclusion setting for minimum-delay analysis only.

-max

With the **-exclude** option, removes the exclusion setting for maximum-delay analysis only.

-all

Removes all crosstalk coupling from all nets.

DESCRIPTION

This command removes delay analysis settings previously specified with the **set_si_delay_analysis** command. Specify the scope of removal by using the same options as the original **set_si_delay_analysis** command.

To get a report on settings currently in effect, use the **report_si_delay_analysis** command.

COMMAND PHASING

The **remove_si_delay_analysis** command must be executed after the **check_design** command but before a path tracing command (such as the **trace_paths** or **extract_model** command).

EXAMPLES

The following command removes the victim exclusion setting previously placed on nets *CLK_NET_**.

```
nt_shell> remove_si_delay_analysis -exclude \
          -victims [get_nets CLK_NET*]
```

The following command removes all previous settings made with **set_si_delay_analysis** commands.

```
nt_shell> remove_si_delay_analysis -all
```

SEE ALSO

```
report_si_delay_analysis(2)
report_si_nets(2)
set_si_delay_analysis(2)
si_enable_analysis(3)
```

remove_si_noise_analysis

Removes the selection exceptions previously set on nets with the **set_si_noise_analysis** command.

SYNTAX

```
status remove_si_noise_analysis
   -exclude | -all
   [-victims vnets]
   [-aggressors anets]
   [-above]
   [-below]
   [-high]
   [-low]
```

Data Types

```
   vnets     list
   anets     list
```

ARGUMENTS

-exclude

Removes the effects of a **set_si_noise_analysis -exclude** command.

-all

With the **-all** option, all noise exceptions are removed from all nets.

-victims *vnets*

With the **-exclude** option, specifies a list of nets from which to remove the victim exclusion setting.

-aggressors *anets*

With the **-exclude** option, specifies a list of nets from which to remove the aggressor exclusion setting.

-above

With the **-exclude** option, removes the exclusion setting related to noise above the stable signal level on the victim net (either high or low).

-below

With the **-exclude** option, removes the exclusion setting related to noise below the stable signal level on the victim net (either high or low).

-high

With the **-exclude** option, removes the exclusion setting related to a high signal on the victim net.

-low

With the **-exclude** option, removes the exclusion setting related to a low signal on the victim net.

DESCRIPTION

This command removes noise analysis settings previously specified with the **set_si_noise_analysis** command. Specify the scope of removal by using the same options as the original **set_si_noise_analysis** command.

To get a report on settings currently in effect, use the **report_si_noise_analysis** command.

COMMAND PHASING

The **remove_si_noise_analysis** command can be executed any time after the **check_design** command.

EXAMPLES

The following command removes the victim exclusion setting previously placed on nets CLK_NET_*.

```
nt_shell> remove_si_noise_analysis -exclude \
          -victims [get_nets CLK_NET*]
```

The following command removes all previous settings made with **set_si_noise_analysis** commands.

```
nt_shell> remove_si_noise_analysis -all
```

SEE ALSO

```
report_si_noise_analysis(2)
report_noise(2)
```

```
report_noise_violation_sources(2)
set_si_noise_analysis(2)
si_enable_noise_analysis(3)
```

remove_steady_state_resistance

Removes steady-state holding resistance values that were previously set for specific model library pins with the **set_steady_state_resistance** command.

SYNTAX

```
status remove_steady_state_resistance
   [-above]
   [-below]
   [-low]
   [-high]
   lib_pin_list
```

Data Types

```
   lib_pin_list          list
```

ARGUMENTS

-above

Removes steady-state resistance for noise above the ground or power rail.

-below

Removes steady-state resistance for noise below the ground or power rail.

-low

Removes steady-state resistance for ground rail noise.

-high

Removes steady-state resistance for power rail noise.

lib_pin_list

A list of model library pins in the current design for which to remove the steady-state resistance values.

DESCRIPTION

The **remove_steady_state_resistance** command removes steady-state holding resistance values that were previously set for specific model library pins with the **set_steady_state_resistance** command.

You can remove steady-state holding resistance values for specific types of noise by using combinations of the **-above**, **-below**, **-low**, and **-high** options.

COMMAND PHASING

The **remove_steady_state_resistance** command can be executed at any time after the **link_design** 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_high(3)
```

remove_storage_node

Removes storage nodes in the current design.

SYNTAX

```
status remove_storage_node
   nets
```

Data Types

```
   nets           collection
```

ARGUMENTS

nets

Removes storage nodes of the specified list of netlists. Provides the list or collection of nets as an argument to the command.

DESCRIPTION

Removes storage nodes in the current design.

EXAMPLES

The following example finds the storage nodes in error conditions, removes the storage nodes which are not marked or recognized as any sequential topologies, then reports the storage nodes again to check that the specified storage nodes have been removed.

```
nt_shell> report_storage_node -errors not_marked -verbose
```

```
****************************************
Report : storage node
Design : top
****************************************

Attributes:
    c - Clocked
    l - Latch
    p - Precharge
    r - Register file


Errors:
    e - Not clocked
    E - Not marked


Storage Nodes      Sub circuits Errors  Attrs Related Storage Nodes
-------------- -------------- ------ ------ ------------------------------
XffnoClock2.L              nand     Ee        {XffnoClock2.L noClockLatchQ2}
noClockLatchQ2              nor     Ee        {XffnoClock2.L noClockLatchQ2}

nt_shell> remove_storage_node "XffnoClock2.L noClockLatchQ2"

nt_shell> report_storage_node -errors all

****************************************
Report : storage node
Design : top
****************************************

Type                            Count
------------------------------ --------
Not marked                          0
Not clocked                         2
```

SEE ALSO

report_channel_connected_block(2)
report_storage_node(2)
report_topology(2)

remove_supply_net

Removes identification of power supply and ground nets in the design.

SYNTAX

```
status remove_supply_net
   net_list
```

Data Types

```
   net_list      list
```

ARGUMENTS

net_list

The list of nets that are not power supply nets or ground nets.

DESCRIPTION

This command removes identification of nets previously identified as power supply nets or ground nets. Nets are automatically identified as power supply nets or ground nets during linking, as determined by the variables **link_vdd_alias** and **link_gnd_alias**. Nets can also be identified by the **set_supply_net** command.

To specify nets in the design that are neither power supply nets nor ground nets, list the net names in the **remove_supply_net** command. For the listed nets, NanoTime sets the net attributes is_supply, is_vdd, and is_ground to false . As a result, NanoTime no longer considers those nets to be power supply or ground nets.

If you use the **remove_supply_net** command after the **match_topology** command and the **topo_match_topology_reset_clock_and_topology** variable is set to **true**, NanoTime repropagates the clock network and rerecognizes topologies at the next **match_topology** or **check_topology** command. If used in lib_topologies, it should be in pre_match_topology phase point.

COMMAND PHASING

The **remove_supply_net** command can only be executed in the "netlist-linked" state. If used after "netlist-linked", NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The following command specifies that the net "g12" is neither a power supply net nor a ground net.

```
nt_shell> remove_supply_net g12
```

SEE ALSO

```
create_lib_topology(2)
set_supply_net(2)
link_gnd_alias(3)
link_vdd_alias(3)
topo_match_topology_reset_clock_and_topology(3)
```

remove_technology

Removes data from the technology library.

SYNTAX

```
status remove_technology
   -techfile file_name | -spice_model file_name
   [-model model_name]
   [-operating_condition op_cond_name]
   [-voltage volts]
   [-temperature temp]
   [-verbose]
```

Data Types

```
   file_name            string
   model_name           string
   op_cond_name         string
   volts                float
   temp                 float
```

ARGUMENTS

-techfile *file_name*

The name of the techfile (TECH: entry) to be removed.

-spice_model *file_name*

The name of the SPICE model (SPM: entry) to be removed.

-model *model_name*

The model or models in the technology to be removed.

-operating_condition *op_cond_name*

Restricts removal of models to those with the specified operating condition name.

-voltage *volts*

Restricts removal of models to those with the specified operating voltage.

-temperature ***temp***

Restricts removal of models to those with the specified operating temperature.

-verbose

Reports entries as they are removed.

DESCRIPTION

The **remove_technology** command removes specified entries from the technology library. You can specify the technology source file names or the model names to remove (or both).

To view a list of technology objects that can be removed, use the **list_technology** command. For example:

```
nt_shell> list_technology
Technology Registry:

Operating
Condition  Volt    Temp   Nmos  Pmos   Source file
---------- ------- ------- ----- -----  ----------------
   typ     1.200  25.000  nc1   pc1    TECH:typ_v3.tech
   typ     1.200  25.000  nch   pch    MOD:netlist
   typ     1.200  25.000  nch   pch    SPM:netlist
1
```

TECH is a label for a techfile-supplied entry. SPM is a label for a SPICE-model entry. MOD is a label for a model generated from a SPICE model template during linking (a "direct read" model).

To remove an entire technology file, use the **-techfile** option:

```
nt_shell> remove_techfile -techfile typ_v3.tech
```

To remove one model from a technology library, specify the model, along with any restricting qualifiers such as operating condition name, temperature, and voltage:

```
nt_shell> remove_techfile -model nc1 -voltage 1.2
```

If you specify multiple qualifiers, only the models that match all qualifiers are removed.

You can remove multiple entries with one command by using wildcard characters in the specified names. Removing models that have been linked generates a warning. Removing models after **check_design** invalidates the design, thus requiring another **check_design** to be performed.

Using the **-verbose** option causes the command to report the names of models being removed.

COMMAND PHASING

The **remove_technology** command can be executed any time before the **check_topology** command. If you use it after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following command removes all techfiles.

```
nt_shell> remove_technology -techfile *
```

The following command removes all SPICE models.

```
nt_shell> remove_technology -spice_model *
```

The following command removes all models named pch in the technology library.

```
nt_shell> remove_technology -model pch
```

The following command removes SPICE models named pch that have certain specified characteristics.

```
nt_shell> remove_technology -model pch \
          -operating_condition wc \
          -temperature 85 -voltage 1.7
```

SEE ALSO

```
list_technology(2)
read_techfile(2)
report_technology(2)
```

remove_timing_check

Removes timing checks.

SYNTAX

```
status remove_timing_check
   {-from from_object
      | -rise_from from_object
      | -fall_from from_object}
   {-to to_object
      | -rise_to to_object
      | -fall_to to_object}
   [-setup | -hold]
   [-of_objects timing_check_objects]
```

Data Types

```
   from_object             list
   to_object               list
   timing_check_objects    list
```

ARGUMENTS

-from *from_object*

Specifies the clock port or clock pin in the current design for the timing check to be removed.

-rise_from *from_object*

Applies only to rising transitions at the clock pin. Similar to the **-from** option.

-fall_from *from_object*

Applies only to falling transitions at the clock pin. Similar to the **-from** option.

-to *to_object*

Specifies the data pin or port in the current design for the timing check to be removed.

-rise_to *to_object*

Applies only to rising signals at the data pin. Similar to the **-to** option.

-fall_to ***to_object***

Applies only to falling signals at the data pin. Similar to the **-to** option.

-setup

Removes the setup check originally specified with the **-setup** option.

-hold

Removes the hold check originally specified with the **-hold** option. If neither of the **-setup** or **-hold** options is specified, both checks are removed.

-of_objects ***timing_check_objects***

Removes timing checks from these timing check objects.

DESCRIPTION

This command removes timing checks, including both automatic topology-based checks and checks previously set with the **create_timing_check** command.

If you are removing checks set with the **create_timing_check** command, the **-from** and **-to** options must exactly match the ones used in the original **create_timing_check** command. This is also true for the **-setup** and **-hold** options, if used in the original **create_timing_check** command.

COMMAND PHASING

If an automatic topology-based check is being removed, the **remove_timing_check** command must be executed in the "design-checked" state (after the **check_design** command). If the timing check being removed was created by the **create_timing_check** command, the **remove_timing_check** command can also be executed in the "topology-checked" state (after the **check_topology** command). In both cases, the **remove_timing_check** command must occur before the **trace_paths**, **extract_model** or **characterize_context** commands. Otherwise, NanoTime transitions back to the "design-checked" state.

EXAMPLES

The following command removes setup timing checks set previously with the **create_timing_check** command.

```
nt_shell> remove_timing_check\
          -rise_from [get_ports CLK]\
          -to [get_ports DATA1] -setup
```

The following command removes hold timing checks set previously with the **create_timing_check** command.

```
nt_shell> remove_timing_check\
          -rise_from [get_ports CLK]\
          -to [get_ports DATA1] -hold
```

SEE ALSO

set_timing_check_attributes(2)
create_timing_check(2)

remove_topology_library

Removes a user library of lib_topology objects.

SYNTAX

```
status remove_topology_library
   name
```

Data Types

```
   name          string
```

ARGUMENTS

name

The name of the user topology library to be removed.

DESCRIPTION

This command removes a user topology library previously read in with the **read_topology_library** command. You cannot remove a topology library if any of its lib_topology objects exist in the design.

COMMAND PHASING

The **read_topology_library** command cannot be used before the **link_design** command. It should be used after the **read_topology_library** command to which it refers.

EXAMPLES

The following command removes the library named user2.

```
nt_shell> remove_topology_library user2
```

SEE ALSO

```
create_topology_library(2)
list_topology_library(2)
match_topology(2)
report_topology_library(2)
```

remove_transistor_drive_factor

Removes transistor drive factors previously set with the **set_transistor_drive_factor** command.

SYNTAX

```
status remove_transistor_drive_factor
   cell_list
```

Data Types

```
   cell_list          list
```

ARGUMENTS

cell_list

The list of transistors from which to remove drive factors.

DESCRIPTION

The **set_transistor_drive_factor** command changes the drive strength of transistors in the design by a specified factor.

To cancel drive factors that has been applied, use the **remove_transistor_drive_factor** command. Specify the list of transistors from which to remove drive factors. This restores the original drive strengths.

COMMAND PHASING

The **remove_transistor_drive_factor** command can be executed any time after the **link_design** command and the **set_transistor_drive_factor** command to which it refers.

EXAMPLES

The following example restores the original drive strength of transistor X3.Mp0.

```
nt_shell> remove_transistor_drive_factor \
          [get_cells X3.Mp0]
```

SEE ALSO

set_transistor_drive_factor(2)

remove_transition_coefficients

Removes transition coefficients previously set with the **set_transition_coefficients** command.

SYNTAX

```
status remove_transition_coefficients
   objects
```

Data Types

```
   objects          list
```

ARGUMENTS

objects

The list of transistor gate pins which select a set of delay arcs which will be affected by this command. This list defines a set of delay calculation trigger pins. The output transitions of delay arcs driven by these trigger pins will no longer be modified.

DESCRIPTION

The **remove_transition_coefficients** command removes transition coefficients previously set with the **set_transition_coefficients** command. You must specify a list of transistor gates from which to remove the coefficients. After running the command, the transition times are calculated normally at the specified transistor gates.

To remove delay coefficients, use the **remove_delay_coefficients** command.

COMMAND PHASING

The **remove_transition_coefficients** command is typically executed in the "netlist-linked" or "topology-checked" state (in other words, between the **link_design** command and the **check_design** command). You cannot use the command before the **link_design** command. If you use the

remove_transition_coefficients command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command removes all transition coefficients previously set on transistor gate pin Xareg.D[53].

```
nt_shell> remove_transition_coefficients \
          [get_pins Xareg.D[53]]
```

SEE ALSO

remove_delay_coefficients(2)
set_delay_coefficients(2)
set_transition_coefficients(2)

rename

Renames or deletes a command.

SYNTAX

```
string rename
   old_name
   new_name
```

ARGUMENTS

old_name

Specifies the current name of the command.

new_name

Specifies the new name of the command.

DESCRIPTION

Renames the *old_name* command so that it is now called *new_name*. If *new_name* is an empty string, then *old_name* is deleted. The *old_name* and *new_name* arguments may include namespace qualifiers (names of containing namespaces). If a command is renamed into a different namespace, future invocations of it will execute in the new namespace. The **rename** command returns an empty string as result.

Note that the **rename** command cannot be used on permanent procedures. Depending on the application, it can be used on all basic builtin commands. In some cases, the application will allow all commands to be renamed.

WARNING: **rename** can have serious consequences if not used correctly. When using **rename** on anything other than a user-defined Tcl procedure, you will be warned. The **rename** command is intended as a means to wrap other commands: that is, the command is replaced by a Tcl procedure which calls the

original. Parts of the application are written as Tcl procedures, and these procedures can use any command. Commands like **puts**, **echo**, **open**, **close**, **source** and many others are often used within the application. Use **rename** with extreme care and at your own risk. Consider using **alias**, Tcl procedures, or a private namespace before using **rename**.

EXAMPLES

This example renames my_proc to my_proc2:

```
prompt> proc my_proc {} {echo "Hello"}
prompt> rename my_proc my_proc2
prompt> my_proc2
Hello
prompt> my_proc
Error: unknown command 'my_proc' (CMD-005).
```

SEE ALSO

define_proc_attributes(2)

report_allow_pin_swap

Report cells with pins that are allowed to be swapped in back annotated parasitics.

SYNTAX

```
status report_allow_pin_swap
```

ARGUMENTS

DESCRIPTION

The **report_allow_pin_swap** command reports cells that have pins which are allowed to be swapped in back annotated parasitics.

RESTRICTIONS

Even if the **parasitics_enable_drain_source_swap** variable is true, globally allowing transistor drain and source pins to be swapped, they will not be reported by **report_allow_pin_swap**.

EXAMPLES

SEE ALSO

set_allow_pin_swap(2)

```
remove_allow_pin_swap(2)
parasitics_enable_drain_source_swap(3)
```

report_analysis_coverage

Reports the coverage of timing checks.

SYNTAX

```
status report_analysis_coverage
   [-status_details status_type]
   [-check_type check_type]
   [-sort_by sort_method]
   [-show_trigger]
   [-compact]
   [-net]
   [-significant_digits digits]
   [-nosplit]
```

Data Types

```
   status_type     string
   check_type      string
   sort_method     string
   digits          integer
```

ARGUMENTS

-status_details *status_type*

Specifies the types of timing checks to include in the report. The allowed values are **removed**, **untested**, **violated**, and **met**. By default, only summary information is shown. Use this option to report individual timing checks.

-check_type *check_type*

Restricts the report to only setup or only hold checks. The allowed values are **setup** and **hold**.

-sort_by

Specifies the method of sorting for the output of the detailed list. The allowed values are **slack**, **name**, and **check_type**. The default is **slack**, which sorts the output first by slack, followed by name, then check_type (setup or hold). An untested or removed check is treated as negative infinity slack.

If you specify **name**, the output is sorted by pin name, followed by slack, then check_type. If you specify **check_type**, the output is sorted by check_type, followed by slack, then pin name.

-show_trigger

Displays the trigger pin for conditional timing checks.

-compact

Replaces the constraint label with an abbreviated endpoint type.

-net

Displays the net of the constrained pin.

-significant_digits ***digits***

Specifies the number of digits to the right of the decimal point. If you do not specify this option, the number of significant digits is determined by the **report_default_significant_digits** variable, whose default is 3.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

DESCRIPTION

The **report_analysis_coverage** command generates a report about the coverage of timing checks in the current design (or the current instance if it is defined). The default report is a summary of checks by type.

For each type of check (setup or hold), the report shows the number and percentage of checks that meet constraints, violate constraints, are untested, and are removed. The report does not show unconstrained output ports, that is, those that have no output delay, maximum delay, or minimum delay. However, the report does show constrained output ports.

You can see more information about the individual timing checks by using the **-status_details** option, which shows details of all checks with the corresponding status. This detail includes a listing of the constrained pin, the related pin, and reason for the check not being analyzed or removed.

The possible reasons for a check not being analyzed are: no_paths, indicating no valid path is found to the checked pin; no_endpoint_clock, indicating no valid clock arrival is found at the related pin; capture_paths_excluded, indicating all capture paths have been excluded when the launch path to the checked pin matched an exclude_reference exception; and constant_disabled, indicating that the net of the checked pin has been forced into a fixed logic state.

The possible reasons for a check being removed are: removed_check, indicating that default timing checks were removed with the **remove_timing_check** or **remove_gated_clock_check** commands; and removed_user_check, indicating that user-defined timing checks were subsequently removed with the **remove_timing_check** or **remove_gated_clock_check** commands.

To use the **report_analysis_coverage** command, you must set the **timing_analysis_coverage** variable to **true** before running the **trace_paths** or **extract_model** command. This variable enables data collection and monitoring of timing checks during path tracing.

To use the full capabilities of the **report_analysis_coverage** and **report_fanin** commands, you must set the **timing_analysis_coverage_fanin** variable to **true**. Note that setting the **timing_analysis_coverage_fanin** variable to **true** also forces the **timing_analysis_coverage** variable to **true**.

Detailed reasons for a check not being analyzed include:

- false_path, indicating that at least one triggered false path exists in the transitive fanin cone
- no_timing_check_path, indicating that at least one triggered no-timing-check path exists in the transitive fanin cone
- data_constant_disabled, indicating that fixed logic propagates through the transitive fanin cone to the checked pin
- clock_constant_disabled, indicating that fixed logic propagates through the transitive fanin cone to the reference pin
- dont_search_thru, indicating that at least one **mark_instance -dont_search_thru_gate** or **mark_instance -dont_search_thru_channel** command affects the transitive fanin cone
- previous_check, indicating that at least one user-defined timing check exists in the transitive fanin cone that was created without the **-continue** option

COMMAND PHASING

The **report_analysis_coverage** command produces useful results only after path tracing is complete.

EXAMPLES

The following example shows a default analysis coverage report.

```
nt_shell> report_analysis_coverage

Type of Check    Total        Met          Violated     Untested
---------------- ------------ ------------ ------------ ------------
setup            10           6 (60%)      0 ( 0%)      4 (40%)
hold             10           6 (60%)      0 ( 0%)      4 (40%)
-------------------------------------------------------------------
Total            20           12 (60%)     0 ( 0%)      8 (40%)
```

The following example shows a detailed analysis coverage report for untested checks sorted by type.

```
nt_shell> report_analysis_coverage -status_details untested -sort_by check_type

Untested reasons:
    NP    - no_paths
    NEC   - no_endpoint_clock
    CPE   - capture_paths_excluded
    CD    - constant_disabled
    UNK   - unknown
    FP    - false_path
    NTC   - no_timing_check_path
    DCD   - data_constant_disabled
```

```
    CCD   - clock_constant_disabled
    DST   - dont_search_thru
    PC    - previous_check

Type of Check    Total        Met          Violated     Untested
---------------- ------------ ------------ ------------ ------------
setup            10           4 (40%)      0 ( 0%)      6 (60%)
hold             10           4 (40%)      0 ( 0%)      6 (60%)
-------------------------------------------------------------------
Total            20           8 (40%)      0 ( 0%)      12 (60%)

                                                  Constrained   Related
type  label                       slack reason            Pin       Pin
----- ------------------------- -------- ------ -------------- ---------
hold  latch hold (transparent)  untested CD     XL1.XI2.MN1.g  XL1.MN1.g
setup latch setup (transparent) untested CD     XL1.XI2.MN1.g  XL1.MN1.g
hold  latch hold (transparent)  untested CD     XL1.XI2.MP1.g  XL1.MN1.g
setup latch setup (transparent) untested CD     XL1.XI2.MP1.g  XL1.MN1.g
hold  latch hold (transparent)  untested NEC    XL4.XI2.MN1.g  XL4.MN1.g
setup latch setup (transparent) untested NEC    XL4.XI2.MN1.g  XL4.MN1.g
hold  latch hold (transparent)  untested NEC    XL4.XI2.MP1.g  XL4.MN1.g
setup latch setup (transparent) untested NEC    XL4.XI2.MP1.g  XL4.MN1.g
hold  set_output_delay check    untested NP     out
hold  set_output_delay check    untested NP     out
setup set_output_delay check    untested NP     out
setup set_output_delay check    untested NP     out
```

The following example shows a detailed analysis coverage report for removed checks sorted by type.

```
nt_shell> report_analysis_coverage -status_details removed -sort_by check_type

Type of Check    Total   Met          Violated     Untested     Removed
---------------- ------- ------------ ------------ ------------ -------
setup            16      10 (62%)     0 ( 0%)      0 ( 0%)      6 (38%)
hold             16      10 (62%)     2 (12%)      0 ( 0%)      4 (25%)
-------------------------------------------------------------------------
Total            32      20 (62%)     2 ( 6%)      0 ( 0%)      10 (31%)

                                                      Constrained       Related
type   label                      removed_(user_)check       Pin     dir      Pin   dir
------ ------------------------- -------------------- ------------ --- --------- ---
hold   latch hold (transparent)   removed_check        XI3.XI2.MP1.g  f  XI3.MN1.g  f
hold   latch hold (transparent)   removed_check        XI4.XI2.MN1.g  r  XI4.MN1.g  f
hold   user specified hold check  removed_user_check   XI7.MN1.g      r  XI7.MN2.g  r
hold   user specified hold check  removed_user_check   XI7.MP1.g      f  XI7.MN2.g  r
setup  latch setup (transparent)  removed_check        XI3.XI2.MP1.g  f  XI3.MN1.g  f
setup  latch setup (transparent)  removed_check        XI4.XI2.MN1.g  r  XI4.MN1.g  f
setup  gated clock setup          removed_check        XI5.MN3.g      r  XI5.MN1.g  r
setup  gated clock setup          removed_check        XI5.MP3.g      f  XI5.MN1.g  r
setup  user specified setup check removed_user_check   XI7.MN1.g      r  XI7.MP2.g  f
setup  user specified setup check removed_user_check   XI7.MP1.g      f  XI7.MP2.g  f
```

SEE ALSO

```
trace_paths(2)
extract_model(2)
report_paths(2)
```

```
report_fanin(2)
set_exclude_reference_path(2)
set_false_path(2)
set_no_timing_check_path(2)
set_case_analysis(2)
mark_instance(2)
create_timing_check(2)
remove_timing_check(2)
remove_gated_clock_check(2)
timing_analysis_coverage(3)
timing_analysis_coverage_fanin(3)
```

report_annotated_device_parameters

Reports device parameters back-annotated on transistors in the current design.

SYNTAX

```
status report_annotated_device_parameters
   [-max_transistors num]
   [-list_annotated]
   [-list_not_annotated]
   [cell_list]
```

Data Types

num	integer
cell_list	list

ARGUMENTS

-max_transistors *num*

Specifies the maximum number of transistors to report for the **-list_annotated** and **-list_not_annotated** options.

-list_annotated

Causes the back-annotated transistors to be listed.

-list_not_annotated

Causes the transistors that are not back-annotated to be listed.

cell_list

Limits the report to the specified cells.

DESCRIPTION

This command generates a report on the number of transistor cells that have been annotated with device parameters by the **read_device_parameters** and **read_parasitics** commands. By default, the report shows how many fingered and non-fingered transistors are annotated and not annotated with device parameters.

Details about transistor annotations can be displayed using the **-list_annotated** or **-list_not_annotated** options. Listing annotated transistors shows the device parameters and annotated values. Listing unannotated transistors shows only a list of transistor cell names. By default, these reports only show 10 transistors. To increase that limit, use the **-max_transistors** option.

You can report device parameters for a specific set of transistor cells by using the *cell_list* option. To see a list of cells, use the **report_cell** command.

COMMAND PHASING

The **report_annotated_device_parameters** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command generates a report on the number of devices that have annotated parameters.

```
nt_shell> report_annotated_device_parameters
...
                                        Transistors
                              Transistors      Not
Type                    Total   Annotated    Annotated
-------------------------------------------------------
Fingered Devices           22        22            0
Non-Fingered Devices    10904     10102          802
-------------------------------------------------------
                        10926     10124          802
1
```

Using the **-list_annotated** option generates the same report, but also lists the annotated values for the first 10 transistors found, as shown in the following example.

```
nt_shell> report_annotated_device_parameters \
          -list_annotated
...

1, mX1/MN0 X1/ZN X1/A gnd gnd nch w=0.89u l=0.13u
as=3.02e-13u ad=5.1e-13u ps=2.21u pd=4.32u

2, mX1/MP0 X1/ZN X1/A vdd vdd pch w=0.89u l=0.13u
```

```
as=3.02e-13u ad=5.1e-13u ps=2.21u pd=4.32u

3, mX1/MP0@1 X1/ZN X1/A vdd vdd pch w=0.89u l=0.13u
as=3.02e-13u ad=5.1e-13u ps=2.21u pd=4.32u
(FINGERED DEVICE)

...
```

SEE ALSO

```
read_device_parameters(2)
read_parasitics(2)
remove_annotated_device_parameters(2)
report_cell(2)
reset_design(2)
write_device_parameters(2)
```

report_annotated_parasitics

Reports net parasitics back-annotated on nets in the current design.

SYNTAX

```
status report_annotated_parasitics
   [-check]
   [-internal_nets]
   [-boundary_nets]
   [-pin_to_pin_nets]
   [-driverless_nets]
   [-loadless_nets]
   [-max_nets num]
   [-list_annotated]
   [-list_annotated_with_lumped_capacitance]
   [-list_not_annotated]
   [net_list]
```

Data Types

```
   num       integer
```

ARGUMENTS

-check

Checks the design to verify that all annotated RC networks are complete. Checks that all fanouts of each net connect through the RC network to each driver of the net.

-internal_nets

Reports only parasitics annotated on internal nets (nets connected only to cell pins). By default, both boundary (port) nets and internal nets are reported.

-boundary_nets

Reports only parasitics annotated on boundary (port) nets. A connection to any port qualifies a net as a boundary net.

-pin_to_pin_nets

Reports only parasitics annotated on nets that have at least one global driver and at least one global

load pin. By default, the option is enabled after check_topology but disabled before check_topology.

-driverless_nets

Reports only parasitics annotated on driverless nets (nets that do not have a global driver). By default, the option is enabled after check_topology but disabled before check_topology.

-loadless_nets

Reports only parasitics annotated on loadless nets (nets that do not have a global load). Note that if there are nets that neither have a global driver nor a global load, those nets are counted as driverless nets and not as loadless nets for the report. By default, the option is enabled after check_topology but disabled before check_topology.

-max_nets ***num***

Specifies the maximum number of nets to report for the **-list_annotated** and **-list_not_annotated** options.

-list_annotated

Reports the back-annotated nets.

-list_annotated_with_lumped_capacitance

Reports the nets that are back-annotated with lumped capacitance.

-list_not_annotated

Reports the nets that are not back-annotated.

net_list

Limits the report to the specified nets.

DESCRIPTION

The **report_annotated_parasitics** command generates a report on the number of nets that have been annotated with parasitics in the current design by the **read_parasitics** command. It can also check the consistency of parasitics after reading ASCII or binary parasitics. The report shows how many nets are annotated with reduced parasitics (pi models) or detailed parasitics (RC networks). Layout-only nets are not included in the report.

Parasitics are associated with global nets. This report counts nets without considering hierarchical crossings. Only one segment per global net is reported.

Details on which nets are annotated or not annotated can be displayed using the **-list_annotated** or **-list_not_annotated** options. Listing annotated nets shows a detailed description of RC networks. Listing unannotated nets shows only a list of net names. By default, these reports only show 10 nets. To increase that limit, use the **-max_nets** option.

You can also report parasitics for a specific set of nets by using the *net_list* option. The **-list_annotated** and **-list_not_annotated** options restrict their scope to the nets in *net_list*.

If the **-check** option is used, the command reports nets with incomplete parasitics.

COMMAND PHASING

The **report_annotated_parasitics** command can be executed at any time after the **link_design** command.

EXAMPLES

The following is an example of an annotated parasitics report.

```
nt_shell> report_annotated_parasitics
 ...
                                  RC       Lumped        Not  Annotation
Net Type             Total  network  Capacitance  Annotated  Incomplete
------------------- ------ -------- ------------ ---------- -----------
Internal nets         2831     1893            0        938           0
 - Pin to pin nets    2399     1891            0        508           0
 - Driverless nets       0        0            0          0           0
 - Loadless nets       432        2            0        430           0
-------------------------------------------------------------------------
Boundary/port nets    1322     1320            0          2           0
 - Pin to pin nets    1320     1320            0          0           0
 - Driverless nets       2        0            0          2           0
 - Loadless nets         0        0            0          0           0
-------------------------------------------------------------------------
                      4153     3213            0        940           0

1
```

Using the **-check** option generates the same report, but also checks and reports unannotated and incompletely annotated nets.

```
nt_shell> report_annotated_parasitics -check
 ...
Error:  pin 'BUS[0]' is missing in the RC annotation
for net 'BUS[0]'.  Ignoring the incomplete RC
annotation. (PARA-006)
Error:  pin 'BUS[1]' is missing in the RC annotation
for net 'BUS[1]'.  Ignoring the incomplete RC annotation. (PARA-006)
 ...
```

SEE ALSO

```
read_parasitics(2)
remove_annotated_parasitics(2)
reset_design(2)
```

report_app_var

Shows the application variables.

SYNTAX

```
string report_app_var
   [-verbose]
   [-only_changed_vars]
   [pattern]
```

Data Types

```
   pattern      string
```

ARGUMENTS

-verbose

Shows detailed information.

-only_changed_vars

Reports only changed variables.

pattern

Reports on variables matching the pattern. The default is "*".

DESCRIPTION

The **report_app_var** command prints information about application variables matching the supplied pattern. By default, all descriptive information for the variable is printed, except for the help text.

If no variables match the pattern, an error is returned. Otherwise, this command returns the empty string.

If the **-verbose** option is used, then the command also prints the help text for the variable. This text is

printed after the variable name and all lines of the help text are prefixed with " #".

The Constraints column can take the following forms:

{val1 ...}

The valid values must belong to the displayed list.

val \<= a

The value must be less than or equal to "a".

val \>= b

The value must be greater than or equal to "b".

b \<= val \<= a

The value must be greater than or equal to "b", and less than or equal to "a".

EXAMPLES

The following are examples of the **report_app_var** command:

```
prompt> report_app_var sh*
Variable                 Value     Type    Default   Constraints
------------------------ --------- ------- --------- --------------------
sh_continue_on_error     false     bool    false
sh_script_stop_severity  none      string  none      {none W E}
 \.\.\.


prompt> report_app_var sh* -verbose
Variable                 Value     Type    Default    Constraints
------------------------ --------- ------- --------- --------------------
sh_continue_on_error     false     bool    false
   # Allows source to continue after an error
sh_script_stop_severity  none      string  none      {none W E}
   # Indicates the error message severity level which would cause
   # a script to stop executing before it completes
 \.\.\.
```

SEE ALSO

```
get_app_var(2)
set_app_var(2)
write_app_var(2)
```

report_arrivals

Reports data arrival times and transition times at specified pins in the design.

SYNTAX

```
status report_arrivals
   [-delay]
   [-transition]
   [-min | -max]
   [-rise | -fall]
   [-clock clock_list]
   [-clock_edge edge_type]
   [-transition_greater_than time_gt]
   [-transition_less_than time_lt]
   [-significant_digits digits]
   [-nosplit]
   -all | pins
```

Data Types

```
   clock_list     list
   edge_type      string
   time_gt        float
   time_lt        float
   digits         integer
   pins           list
```

ARGUMENTS

-delay

Causes only the arrival times to be reported.

-transition

Reports slopes based either on the net-specific **set_measurements_thresholds** slew* threshold settings or on the general rc_slew* threshold variable settings. The default rc_slew* settings are 10 to 90 percent.

-min

Reports only the minimum delays and minimum transitions.

-max

Reports only the maximum delays and maximum transitions.

-rise

Reports only rising transitions and rising delays at the specified pins.

-fall

Reports only falling transitions and falling delays at the specified pins.

-clock ***clock_list***

Reports only the arrivals with respect to the specified clocks. In the absence of this option, all clocks are considered.

-clock_edge ***edge_type***

Reports only the arrivals with respect to the specified edge type of the clock specified with the **-clock** option. Valid settings are **rise** and **fall**.

-transition_greater_than ***time_gt***

Reports only the transitions having a transition time greater than the specified time value. This option can be used with the **-all** option to get a report of the longest transitions.

-transition_less_than ***time_lt***

Reports only the transitions having a transition time less than the specified time value. This option can be used with the **-all** option to get a report of the shortest transitions.

-significant_digits ***digits***

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable, whose default is 3.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

-all

Causes all timing points in the path database that meet the command criteria to be reported.

pins

Specifies the pins to be reported.

DESCRIPTION

The **report_arrivals** command reports the arrival times and transition times of transitions occurring at pins along data paths. This is an example of a report:

```
Pin/Port      Logic     Delay   Transition Phase Clock
------------ --------- ------- ---------- ----- ------
Xlc5.Mn1.G    min_rise  0.145   0.045      rise  MCLK
```

To use this command, you must first set the **timing_save_pin_arrival_and_transition** variable to **true** before you run the **trace_paths** command. By default, this variable is set to **false**, which means that the arrival time and transition time information is not saved during path tracing and the **report_arrivals** command does not return any information. When the variable is set to **true**, NanoTime saves the worst-case minimum and maximum delays and transition times at each channel-connected block (CCB) boundary. Only these values can be reported.

If the variable is set to **true**, you can use the **report_arrivals** command after path tracing is complete to get a report on the arrival times and transition times of specified pins or all pins in the design. The **-rise**, **-fall**, **-min**, **-max**, **-clock**, and **-clock_edge** options restrict the scope of the report to certain edges, minimum or maximum delays or transition times, or specified clocks or clock edges.

To limit the report to transition times that are greater than or less than a specific value, use the **-transition_greater_than** or **-transition_less_than** option.

By default, the reported arrival times include the launch clock arrival times. 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, the arrival at the pin is reported to occur at time 5.5. To exclude the clock arrival time and report only the clock-to-pin delay, set the **timing_arrival_shift_reference** variable to **true** before you execute the **trace_paths** command.

If the **timing_save_wire_delay** command is set to **true**, arrival times are also available at the driver pins of an RC network on a channel-connected block boundary net.

COMMAND PHASING

The **report_arrivals** command produces useful results only after path tracing is complete.

EXAMPLES

The following report shows the worst-case arrival times and transitions time at pin Xlc5.Mn1.G.

```
nt_shell> report_arrivals Xlc5.Mn1.G
 ...
Pin/Port      Logic     Delay    Transition Phase Clock
------------ --------- -------- ---------- ----- ------
Xlc5.Mn1.G    min_rise  0.145    0.045      rise  MCLK
Xlc5.Mn1.G    max_rise  0.145    0.045      rise  MCLK
```

The following report shows the worst-case minimum delay rising transitions at all pins in the design where the transition time is greater than 0.80 time units.

```
nt_shell> report_arrivals -min -rise \
          -transition_greater_than 0.80 -all
 ...
```

```
Pin/Port        Logic     Delay   Transition Phase Clock
--------------- --------- ------- ---------- ----- ------
Xlc30.X5.Mn0.G  min_rise  1.271   1.021      rise  clk2
Xlc51.X5.Mn0.G  min_rise  1.331   1.030      rise  clk3
Xlc59.X5.Mn0.G  min_rise  1.341   1.030      rise  clk3
Xlc5.X5.Mn0.G   min_rise  1.230   1.023      rise  clk2
 ...
```

The following report shows the worst-case arrival times and transition times at all pins in the design and writes the report to a file.

```
nt_shell> report_arrivals -all > arrivals.txt
```

SEE ALSO

```
trace_paths(2)
timing_arrival_shift_reference(3)
timing_save_wire_delay(3)
timing_save_pin_arrival_and_transition(3)
```

report_attribute

Reports the attribute values of one or more objects in the design.

SYNTAX

```
status report_attribute
   [-application]
   [-class class_name]
   [-nosplit]
   object_spec
```

Data Types

```
   class_name        string
   object_spec       list
```

ARGUMENTS

-application

Includes all attributes defined by NanoTime.

-class *class_name*

If *object_spec* is a name, this is its class.

-nosplit

Prevents the splitting of lines when columns overflow.

object_spec

A single object from which to get the attribute value. The *object_spec* must be either a collection of exactly one object, or a name which is combined with the **-class** option to find the object.

DESCRIPTION

The **report_attribute** command generates a report of the values of attributes associated with specified objects in the design. The **-application** option is required.

For application attributes which are of the type *collection*, the name of the first object in the collection is displayed.

If *object_spec* is a name, the **-class** option must also be used. Valid values for the **-class** option are:

capture_window_edge
cell
clock
design
launch_window_edge
lib
lib_cell
lib_pin
lib_topology
model_clock_domain
model_delay_arc
net
parasitic_device
path_arc
pin
port
simulation
timing_check
timing_path
timing_point
topology

To get a list of available attributes, use the **list_attributes -application** command.

To get the value of a single attribute associated with an object, use the **get_attribute** command.

COMMAND PHASING

The **report_attribute** command is not phase-restricted.

EXAMPLES

The following command reports all attributes that are associated with the cell named Xxor7. The attributes provide information about the cell such as the its function and number of pins.

```
nt_shell> report_attribute -application [get_cells Xxor7]

...
Design   Object    Type      Attribute Name    Value
-------------------------------------------------

ALU      Xxor7     string    object_class      cell
ALU      Xxor7     string    base_name         Xxor7
ALU      Xxor7     string    full_name         Xxor7
ALU      Xxor7     string    ref_name          buf2
ALU      Xxor7     boolean   is_transistor     false
ALU      Xxor7     boolean   is_nmos           false
ALU      Xxor7     boolean   is_pmos           false
```

```
ALU     Xxor7    boolean   is_resistor       false
ALU     Xxor7    boolean   is_capacitor      false
ALU     Xxor7    boolean   is_timing_model   false
ALU     Xxor7    boolean   is_hierarchical   true
ALU     Xxor7    int       number_of_pins    2
ALU     Xxor7    boolean   is_clock_gate     false
ALU     Xxor7    boolean   is_feedback       false
ALU     Xxor7    boolean   is_latch          false
ALU     Xxor7    boolean   is_mux            false
ALU     Xxor7    boolean   is_precharge      false
ALU     Xxor7    boolean   is_ram            false
ALU     Xxor7    boolean   is_tgate          false
ALU     Xxor7    boolean   is_weak_pullup    false
ALU     Xxor7    boolean   is_xor            false
```

SEE ALSO

```
get_attribute(2)
list_attributes(2)
```

report_bitline

Generates a report on the bitlines associated with a memory in the design.

SYNTAX

```
status report_bitline
   [-bitcells]
   [-nosplit]
   [net_list]
```

Data Types

```
   net_list     list
```

ARGUMENTS

-bitcells

Reports the bitcells on the bitlines.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

net_list

A list of bitlines to be reported. Each element in this list is either a collection of bitline nets or a pattern matching the net names. Without this option, all bitlines are reported.

DESCRIPTION

This command displays information about bitlines in the current design. The types of bitline information reported by the command depends on the options used. By default, the command produces a list of

bitlines items and their values.

```
Items:
    Memory             associated memory name
    Complement Net     net
    Precharge Pin      pin
    Precharge Net      net
    Precharge Txs      list of transistors
    Num Bit Cells      number of bit cells

    Read Mux Select    list of mux select pins
    Sense Amp Input    net
    Sense Amp Enable   pin
    S-Amp Prech Pin    pin
    S-Amp Prech Net    net
    Write Out Nets     list of nets
```

Using the **-bitcells** option adds extra rows in the items list for the bitcell name and wordline for each bit cell associated with the bitline.

```
Items:
    Wordline   0   wordline net name
    Bit-cell   0   list of cells containing bitline and its complement
```

COMMAND PHASING

The **report_bitline** command can be executed at any time after the **link_design** command.

EXAMPLES

This example shows the default bitline report.

```
nt_shell> report_bitline Xmem/Xmem0/bit0
 ...
Attributes:
    s - selected

Memory bitline 'Xmem/Xmem0/bit0':

Item              Value
----------------- - ------------------------------------------------------
Memory            read_mem
Complement Net    Xmem/Xmem0/bitb0
Precharge Pin     Xmem/Xmem0/Xbitpre0/Mp0/g
Precharge Net     Xmem/prech
Precharge Txs     {Xmem/Xmem0/Xbitpre0/Mp0 Xmem/Xmem0/Xbitpre0/Mp1 ...}
Num Bit Cells     2
Read Mux Select   {Xmem/Xmem0/Xrdmux/Mp0/g Xmem/Xmem0/Xrdmux/Mp1/g}
Sense Amp Input   Xmem/Xmem0/data0
Sense Amp Enable  Xmem/Xmem0/Xsenseamp_xcoupled-inv-nen-pre/Mnsaen/g
S-Amp Prech Pin   Xmem/Xmem0/Xsenseamp_xcoupled-inv-nen-pre/Xpre/Mp0/g
S-Amp Prech Net   Xmem/prechsamp
Write Out Nets    Xmem/Xmem0/bit0
```

SEE ALSO

```
create_memory(2)
get_memory(2)
report_memory(2)
report_bitline(2)
```

report_case_analysis

Reports case analysis settings on ports and pins.

SYNTAX

```
status report_case_analysis
   [-all]
   [-nosplit]
```

ARGUMENTS

-all

Reports the pins upon which you have set case analysis values and reports the built-in constant pins of the design that are considered startpoints of logic constant propagation.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

DESCRIPTION

The **report_case_analysis** command reports case analysis settings on ports and pins previously set with the **set_case_analysis** command. The report does not include information about the propagation of those settings into the design.

COMMAND PHASING

The **report_case_analysis** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports all case analysis values that have been set.

```
nt_shell> report_case_analysis
 ...
Case  Net
----  -------------------------
one   selA
zero  selB
```

SEE ALSO

remove_case_analysis(2)
set_case_analysis(2)

report_cell

Reports cell information.

SYNTAX

```
status report_cell
   [-connections]
   [-verbose]
   [-nosplit]
   [-top_net_of_top_hierarchical_group]
   [-significant_digits digits]
   [cell_list]
```

Data Types

```
   digits           integer
   cell_list        list
```

ARGUMENTS

-connections

Displays the pins of the cell and the nets to which they connect.

-verbose

Displays detailed information about the nets to which the cell pins are connected: for each input pin, the net name and driver pins on that net; and for each output pin, the net name and load pins on that net. The **-verbose** option can be used only with the **-connections** option.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

-top_net_of_top_hierarchical_group

Reports the equivalent highest level hierarchical net names. By default, the **-connections** option reports the names of the connected nets at the same hierarchical level as the cell. With this option, when more than one hierarchical net of the same group is connected to the cell (local nets at various

hierarchical levels of the same physical net), only the net closest to the top of the hierarchy is reported. In the case of multiple nets at the same level, only the first net found is reported.

-significant_digits ***digits***

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant more digits is determined by the **report_default_significant_digits** variable, whose default is 3.

cell_list

Specifies the cells to be reported. In the absence of this option, the command lists all cells in the current instance.

DESCRIPTION

This command lists all cells in the current instance or displays information about the nets and connected pins of specified cells. Use the **-connections** option to get a list of cell pins and connected pins, or **-connections -verbose** to also get information about pins of other cells connected to the same nets.

If you set the current instance with the **current_instance** command, the **report_cell** command reports the cells within the current instance. Otherwise, it reports on cells in the current design.

COMMAND PHASING

The **report_cell** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command generates a list of all cells in the current design.

```
nt_shell> report_cell
Attributes:
    t - transistor
    h - hierarchical

Cell                Reference       Library     Attributes
------------------- --------------- ----------- ----------
Xaddsub             addsub_unit                 h
Xand0               buf2                        h
Xand1               buf2                        h
Xand2               buf2                        h

...

Xxor6               buf2                        h
Xxor7               buf2                        h
```

```
------------------------------------------------------------
Total 205 cells
```

The following command reports the pin names and net connections of cell Xxor7.

```
nt_shell> report_cell -connections Xxor7
...
Connections for cell 'Xxor7':
    Reference:      buf2
    Hierarchical:   TRUE

    Inout Pins      Net
    --------------- ---------------
    A               xor
    Z               xor7
```

The following command reports the pin names, net connections, and pin of other cells connected to those nets, for cell Xxor7.

```
nt_shell> report_cell -connections -verbose Xxor7
...
Connections for cell 'Xxor7':
  Reference:      buf2
  Hierarchical:   TRUE


  Inout Pins    Net      Other Pins   Pin Type
  ------------- -------- ------------ ----------------------
  A             xor      Xxor7.A      Inout Pin (buf2)
  Z             xor7     Xlc63.xor    Inout Pin (logic_cell)
```

SEE ALSO

current_design(2)
current_instance(2)
report_design(2)
report_hierarchy(2)
report_net(2)

report_channel_connected_block

Reports channel-connected blocks (CCBs) in the current design.

SYNTAX

```
status report_channel_connected_block
   [-verbose]
   [-input_connection]
   [-output_connection]
   [-nosplit]
   [nets]
```

Data Types

```
   nets          collection
```

ARGUMENTS

-verbose

Displays nets and devices in the channel.

-input_connection

Displays the CCB inputs which are the nets driving the gate pins of the CCB devices. Also displays the fanin CCBs which drive the inputs.

-output_connection

Displays the CCB outputs and the fanout CCBs which are driven by the outputs.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

nets

Limits channel-connected block reporting to the specified list of nets. Provide the list or collection of nets as an argument to the command.

DESCRIPTION

Reports channel-connected blocks in the current design. Without any options, the command displays a summary of every CCB including the number of nets in the channel, the number of devices and nets in the block, the number of inputs, the number of outputs, an attribute describing the type of circuit, and a label.

Use the **-verbose** option to get detailed information of the CCB. Use the **-input_connection** and **-output_connection** options to get more information about fanout and fanin nets for specific channel-connected blocks. Limit the CCB reporting by specifying a collection of nets instead of reporting all nets.

COMMAND PHASING

The **report_channel_connected_block** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example shows the summary report of all CCBs in the current design.

```
nt_shell> report_channel_connected_block

****************************************
Report : channel connected block
Design : top
****************************************

Attributes:
    b - CCB with Feedbacks
    c - Clocked CCB
    g - Clock gate
    f - Forced clock
    l - Latch
    p - Precharge
    r - Register file
    s - Stopped clock

CCB ID :     A net in the CCB which is the representative of the CCB
CCB Inputs : The nets which drive the gate pins of the CCB devices
CCB Outputs: The nets in the CCB which drive the inputs of other CCBs

                     #CCB   #CCB   #CCB     #CCB
Nets              Devices   Nets Inputs  Outputs  Attrs CCB ID
--------------- -------- ------ ------ -------- ------ ---------------
CK1bar                 2      1      1        1      c CK1bar
CKbar                  2      1      1        1      c CKbar
Dbar                   2      1      1        1        Dbar
Xffmaster.CKN          2      1      1        1      c Xffmaster.CKN
Xffmaster.Dbar         6      2      4        1     cl Xffmaster.Dbar
....
resetBar               2      1      1        1        resetBar
```

```
    slaveQ                2      1      1        1        slaveQ

    Total CCBs : 26
```

SEE ALSO

current_design(2)
report_net(2)

report_clock

Reports information about clocks in the design.

SYNTAX

```
status report_clock
   [-attributes]
   [-skew]
   [-nosplit]
   [clock_names]
```

Data Types

```
   clock_names          list
```

ARGUMENTS

-attributes

Shows clock attributes and provides a list of all the clocks in the current design. This is the default report.

-skew

Reports clock latency (source and network latency) and uncertainty information set on the design by the **set_clock_latency** and **set_clock_uncertainty** commands.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

clock_names

The list of clocks to be reported. If this option is not used, all clocks are reported.

DESCRIPTION

The **report_clock** command provides information on clocks in the design. Clocks are created by the **create_clock** and **create_generated_clock** commands. You can generate a clock attribute report (the default) or a skew report (using the **-skew** option).

A clock attribute report shows the clock name, period, edge times, clock attribute (type), and clock source for each clock in the design. For example:

```
Attributes:
    p - Propagated clock
    G - Generated clock
    H - Pulse clock, high
    L - Pulse clock, low
    D - Differential clock

Clock      Period Waveform       Attrs  Sources
---------- ------ -------------- ------ -------
CK         3.000  {0.000 1.500}  p      ck
CK1        4.000  {0.000 2.000}  p      cka
```

The numbers in the Waveform column indicates the times at which the rising and falling edges of the clock occur.

The letters in the Attrs column indicate the type of clock (propagated, generated, pulse clock high, pulse clock low, or differential). A propagated clock uses calculated rather than ideal delays. A pulse clock is a clock whose leading and trailing edges are triggered by the same event, but with different delays from the common triggering event. For more information, see the man page for the **create_generated_clock** command.

The Sources column indicates the source object in the design specified at the time of clock creation.

To get a report on clock latency and uncertainty for an ideal (not propagated) clock, use the **-skew** option of the **report_clock** command. For more information about latency and uncertainty, see the man pages for the **set_clock_latency** and **set_clock_uncertainty** commands.

COMMAND PHASING

The **report_clock** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example demonstrates the creation and reporting of a clock.

```
nt_shell> create_clock -period 20.0\
          -name ck2 [get_ports {CLK}]
```

```
nt_shell> report_clock

...
Attributes:
    p - Propagated clock
    G - Generated clock
    H - Pulse clock, high
    L - Pulse clock, low

Clock   Period Waveform          Attrs Sources
-----------------------------------------------
ck2     20.000 {0.000 10.000}    p     clk2
```

SEE ALSO

all_clocks(2)
create_clock(2)
create_generated_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)

report_clock_arrivals

Reports clock arrival times at sequential device pins or at points along clock trees.

SYNTAX

```
status report_clock_arrivals
   [-tree]
   [-min]
   [-max]
   [-rise]
   [-fall]
   [-nets]
   [-significant_digits digits]
   [-nosplit]
   [clock_list]
```

Data Types

digits	integer
clock_list	list

ARGUMENTS

-tree

Causes reporting of the full clock propagation tree. In the absence of this option, the command reports the clock arrival times at the sequential device pins in the design.

-min

Shows only the shortest path arrivals for a full-tree report.

-max

Shows only the longest path arrivals for a full-tree report.

-rise

Shows only the arrivals from the rising edge of the clock for a full-tree report.

-fall

Shows only the arrivals from the falling edge of the clock for a full-tree report.

-nets

Shows the net name at each clock propagation timing point for a full-tree report, rather than the pin name.

-significant_digits ***digits***

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable, whose default is 3.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

clock_list

Specifies the nets (without the **-tree** option) or clocks (with the **-tree** option) that are to be reported. If no objects are specified, the arrival information is reported for all nets or all clocks.

DESCRIPTION

The **report_clock_arrivals** command reports the arrival times of clock signals at sequential device pins in the design or along intermediate points of a full clock tree.

Any time after the **check_design** command has been executed, you can use the **report_clock_arrivals** command to view a list of the clock nets for debugging the clock network, even before the **trace_paths** command has been run. However, to see the actual arrival times in the report, you must execute the **trace_paths** command first.repor

In the **report_clock_arrivals** command, use the **-tree** option to get a report on a full clock tree, or omit the **-tree** option to get a report on arrival times at all sequential device pins.

Without the **-tree** option, the report looks like this:

```
...
Attributes:
    H - Pulse clock with high pulse
    L - Pulse clock with low pulse
    D - Differential clock
    les
Net             Min Rise  Min Fall  Max Rise  Max Fall Period Attrs Clock
--------------- --------- --------- --------- --------- ------ ----- -----
Xsreg.ck10         3.549     0.322     3.628     0.423  6.000       clk2n
Xbreg.ck10         4.052     0.830     4.134     0.937  6.000       clk2n
Xareg.ck10         4.160     1.013     4.245     1.095  6.000       clk2n
 ...
```

The report lists the sequential device pins and the Min Rise, Min Fall, Max Rise, and Max Fall arrival times at

each pin. Min Fall means the earliest (minimum-delay) arrival time at the device pin with respect to the rising edge of the clock at the clock source pin.

With the **-tree** option, the report looks like this:

```
Max trees for clock 'clk1'

0.000 r clk1 (Clock source pin)
|- 0.000 r Xreg15.X1.Mn0.g(nand2) (via net)
|  |- 0.048 f Xreg15.X2.Mp0.g(inv2) (via clock_gate)
|  |  |- 0.078 r Xreg15.Mn4.g(dff) (via inverter) (checked pin)
|  |  `- 0.078 r Xreg15.Mp0.g(dff) (via inverter) (checked pin)
|  |- 0.048 f Xreg15.Mp4.g(dff) (via clock_gate) (checked pin)
|  `- 0.048 f Xreg15.Mn0.g(dff) (via clock_gate) (checked pin)
|- 0.000 r Xreg14.X1.Mn0.g(nand2) (via net)
|  |- 0.048 f Xreg14.X2.Mp0.g(inv2) (via clock_gate)
|  |  |- 0.078 r Xreg14.Mn4.g(dff) (via inverter) (checked pin)
|  |  `- 0.078 r Xreg14.Mp0.g(dff) (via inverter) (checked pin)
|  |- 0.048 f Xreg14.Mp4.g(dff) (via clock_gate) (checked pin)
|  `- 0.048 f Xreg14.Mn0.g(dff) (via clock_gate) (checked pin)
 ...
```

This report shows the full clock network using a tree representation. Each node in the tree, represented by one line in the report, is a timing point in the clock tree. Multiple points propagated from a given point are shown indented below that point.

At each timing point, the line in the report shows the following information:

```
1. Arrival time (delay from clock source)

2. Edge direction at clock source:
    r = rising
    f = falling

3. Timing point name (pin or net name)

4. Method of propagation from previous point:
    via net = connection from an input port
    via inverter = propagation through an inverter
    via clock_gate = propagation across a clock gate
    via force_propagation = mark_clock_network setting
    via unknown gate type = other situation

5. If a clock endpoint, the reason propagated stopped:
    checked pin = sequential device clock pin
    generated clock pin = create_generated_clock pin
    stop_propagation = mark_clock_network setting
    end of clock network = no fanout, output port
    false path, logic blocking, other = other situation
```

When you use the **-tree** option, you can use the **-min** or **-max** and **-rise** or **-fall** options to restrict the scope of the report. You can also specify which clocks to report. Use the **-nets** option if you want the report to show net names rather than pin names.

Some clock network circuits contain reconvergent paths, where two paths diverge and rejoin. This is condition is indicated in the report by the word RECONVERGENT.

The report indicates that a clock is considered a pulse clock at a given timing point by displaying either PULSE HIGH or PULSE LOW. For information about pulse clocks, see the man page for the **create_clock**

command.

COMMAND PHASING

The **report_clock_arrivals** command can be executed at any time after the **check_design** command. However, you must run the **trace_paths** command first to obtain arrivals.

EXAMPLES

The following report shows the full-tree, maximum-delay arrivals resulting from the rising edge of clk2.

```
nt_shell> report_clock_arrivals -tree -max -rise \
          [get_clocks clk1]

Max trees for clock 'clk1'

0.000 r clk1 (Clock source pin)
`- 0.000 r xck1.mn1.g(inv) (via net)
   |- 1.000 f xck2.mp1.g(inv) (via inverter)
   |  `- 2.000 r xl1.mn1.g(lat) (via inverter) (checked pin)
   |- 1.000 f xck3.mp1.g(inv) (via inverter)
   |  `- 2.000 r xl2.mn1.g(lat) (via inverter) (checked pin)
   `- 1.000 f xl3.mn1.g(lat) (via inverter) (checked pin)
```

SEE ALSO

```
create_clock(2)
mark_clock_network(2)
trace_paths(2)
report_default_significant_digits(3)
```

report_clock_gating_check

Displays clock gating check information about specified pins.

SYNTAX

```
status report_clock_gating_check
   [-significant_digits digits]
   [-nosplit]
   [clock_pin_list]
```

Data Types

```
   digits               integer
   clock_pin_list       list
```

ARGUMENTS

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable, whose default is 3.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

clock_pin_list

Specifies a list of clocks or pins to use to select the checks to report.

DESCRIPTION

The **report_clock_gating_check** command displays the following information for each clock gating

check: the enable pin and edge direction, the output (clock) pin and edge direction, the type of the check, and the margin limit.

COMMAND PHASING

The **report_clock_gating_check** command can be executed at any time after the **check_design** command.

EXAMPLES

The following example shows the reporting of all clock gating checks.

```
nt_shell> report_clock_gating_check
*************************************
Report : clock gating check
Design : top
Version: V-2003.12
Date   : Mon Jul 19 11:37:56 2004
*************************************

Enable Pin                Output Pin                Type     Limit
-----------------------------------------------------------------------
r xck1.mn2.g              f out1                    setup     0.00
f xck1.mn2.g              r out1                    hold      0.00
r xck1.mp2.g              f out1                    setup     0.00
f xck1.mp2.g              r out1                    hold      0.00
f xck2.mn2.g              r out2                    setup     0.00
r xck2.mn2.g              f out2                    hold      0.00
f xck2.mp2.g              r out2                    setup     0.00
r xck2.mp2.g              f out2                    hold      0.00
1
```

SEE ALSO

```
create_gated_clock_timing_check(2)
mark_clock_gate(2)
report_constraint(2)
report_default_significant_digits(3)
```

report_clock_network

Reports the clock network in the current design.

SYNTAX

```
status report_clock_network
   [-from net_or_clock]
   [-verbose]
   [-errors]
   [-nosplit]
   [net]
```

Data Types

```
   net_or_clock      string
   net               string
```

ARGUMENTS

-from net_or_clock

Displays the clock network from the specified point.

-verbose

Displays detail information about the clock network, the error conditions, or the clock path.

-errors

Shows error conditions within the clock network.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

net

Displays clock paths or possible clock paths to this net.

DESCRIPTION

Without any options, the **report_clock_network** command displays a summary report of the clock network in the current design. The summary includes the number of clock nets, the number of clock gates, and the number of valid and invalid endpoints.

Use the **-errors** option to get the error conditions within the clock network. Limit the clock network reporting by specifying the source point with the **-from** option. Use the **-verbose** option to get detailed information.

COMMAND PHASING

The **report_clock_network** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports the clock network from clock net CK.

```
nt_shell> report_clock_network -from [get_net "CK"] -verbose

**************************************
Report : clock network
Design : top
**************************************

Clock gate types:
    p - Pulse Generator
    s - Pulse Shaper


Clock network:            CK
Number of clock nets:     7
Number of standard clock gates: 1
Number of pulse generator: 0
Number of pulse shaper:   0

Clock Nets
---------------
CK
CKbar
ffmaster.CKN
ffslave.CKN
latch.CKN
gated_clock
out0

Clock Gate Nets                Clock Gate Type
------------------------------ ---------------
gated_clock

1
```

The following command reports the errors in clock network CK.

```
nt_shell> report_clock_network -errors -from [get_net "CK"] -verbose

****************************************
Report : clock network
Design : top
****************************************

Clock network:            CK
Number of invalid endpoints :  6

Invalid end point pins                              Reason
------------------------------ ------------------------------
gated_clock.M0.g                      exceed clock gate limit
gated_clock.M2.g                      exceed clock gate limit
latch.fer.MN0.g                  missing sequential topology
latch.fer.MP0.g                  missing sequential topology
unknown_gated_clock.Mn.g                  missing clock gate
unknown_gated_clock.Mp.g                  missing clock gate


Missing clock nets:       5

Missing clock nets   Reason
-------------------- --------------------
precharge.ck         precharge without clock
latch.ck             latch without clock
out0                 stopped clock net
out1                 forced clock net
set                  requires clock

1
```

SEE ALSO

```
report_channel_connected_block(2)
report_clock(2)
report_storage_node(2)
report_topology(2)
set_requires_clock(2)
remove_requires_clock(2)
```

report_constraint

Displays constraint-related information about a design.

SYNTAX

```
status report_constraint
   [-all_violators]
   [-verbose]
   [-path_type format]
   [-max_delay]
   [-min_delay]
   [-max_capacitance]
   [-min_capacitance]
   [-max_transition]
   [-min_transition]
   [-min_pulse_width]
   [-recovery]
   [-removal]
   [-max_skew]
   [-clock_gating_setup]
   [-clock_gating_hold]
   [-stopped_clock_setup]
   [-stopped_clock_hold]
   [-data_setup]
   [-data_hold]
   [-clock_setup]
   [-clock_hold]
   [-non_transparent_setup]
   [-non_transparent_hold]
   [-nets]
   [-significant_digits digits]
   [-nosplit]
```

Data Types

```
format     string
digits     integer
```

ARGUMENTS

-all_violators

Displays a summary showing the worst violation per endpoint of each violated constraint in the current

design. Multiple violations for a given constraint are listed in order, from largest to smallest violation.

-verbose

Causes the report to show detailed information on each constraint violation.

-path_type ***format***

Specifies the format for the path report. Allowed values are *slack_only* (the default) and *end*. This option has an effect only if the **-verbose** option is not used. If *slack_only* is specified, the report displays only endpoint slacks. If *end* is specified, the report has a column format that shows one line for each path showing the data arrival time, data required time, and slack.

-max_delay

Causes only maximum-delay (setup) constraints to be reported.

-min_delay

Causes only minimum-delay (hold) constraints to be reported.

-max_capacitance

Causes only the maximum capacitance constraint to be reported. This constraint limits the total capacitance allowed on each net. The **-max_capacitance** option by itself displays the maximum capacitance cost (the sum of all max_capacitance violations). To see details about the worst violator, use the **-verbose** option together with the **-max_capacitance** option. You can also use the **-all_violators** option for information on all maximum capacitance violations.

-min_capacitance

Causes only the minimum capacitance constraint to be reported.

-max_transition

Causes only the maximum transition time constraint to be reported. This constraint specifies the maximum allowable transition time on ports and pins.

-min_transition

Causes only the minimum transition time constraint to be reported. This constraint specifies the minimum allowable transition time on ports and pins.

-min_pulse_width

Causes only the minimum pulse width constraint to be reported. This constraint specifies the minimum allowable pulse width, either high or low, for a signal at any clock pin or pin in a clock network.

-recovery

Causes only the recovery constraint to be reported. This constraint specifies the minimum allowable time between the control pin transition to the inactive state, and the active edge of the synchronous clock signal. This time is from the control signal going inactive to the clock edge that latches data in. The asynchronous control signal must remain constant during this time to prevent an incorrect value at the outputs.

-removal

Causes only the removal constraint to be reported. This constraint specifies the minimum allowable time

between the clock pin inactive edge, while the asynchronous pin is active, to the inactive edge of the same asynchronous control pin.

-max_skew

Causes only the maximum skew constraint to be reported. This constraint specifies the maximum separation time allowed between two clock signals.

-clock_gating_setup

Causes only the clock-gating setup constraints to be reported. This constraint specifies the minimum setup time between a clock and a signal controlling the gating of that clock.

-clock_gating_hold

Causes only the clock-gating hold constraints to be reported. This constraint specifies the minimum hold time between a clock and a signal controlling the gating of that clock.

-stopped_clock_setup

Causes only the stopped-clock setup constraints to be reported. This constraint is a setup check performed where the **mark_clock_network -stop_propagation -add_setup** command has been used.

-stopped_clock_hold

Causes only the stopped-clock hold constraints to be reported. This constraint is a hold check performed where the **mark_clock_network -stop_propagation -add_hold** command has been used.

-data_setup

Causes only the data setup constraints defined by the **set_data_check** command to be reported. This constraint specifies the minimum setup time between a "from" object and a "to" object in the design.

-data_hold

Causes only the data hold constraints defined by the **set_data_check** command to be reported. This constraint specifies the minimum hold time between a "from" object and a "to" object in the design.

-clock_setup

Causes only the clock-to-clock setup constraints to be reported. This type of constraint is a setup check between two clock pins. It can be a contention check between the precharge and evaluate clocks at a precharge structure or a setup check preformed where the **create_timing_check** command has identified a clock pin in both the "from" object and "to" object.

-clock_hold

Causes only the clock-to-clock hold constraints to be reported. This type of constraint is a hold check between two clock pins. It can be a contention check between the precharge and evaluate clocks at a precharge structure or a hold check performed where the **create_timing_check** command has identified a clock pin in both the "from" object and "to" object.

-non_transparent_setup

Causes only the nontransparent setup constraints to be reported, where the **set_non_transparent** command has been used to disable transparent checking.

-non_transparent_hold

Causes only the nontransparent hold constraints to be reported, where the **set_non_transparent** command has been used to disable transparent checking.

-nets

Includes the net names.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

DESCRIPTION

The **report_constraint** command displays a report on the success or failure of the design in meet its timing constraints and design rule constraints. The **report_constraint** command can be used only after path tracing has been completed with the **trace_paths** command.

The contents of the report depend on the options used in the command. By default, without any options, the command generates a summary report that lists the constraints evaluated by the **trace_paths** command, and for each constraint, the amount of the worst violation or the smallest amount by which the constraint was met. For example:

```
Constraint                      Slack
------------------------------ -------- --------
max_delay/setup                -0.653 (VIOLATED)
min_delay/hold                  0.102 (MET)
clock_tree_pulse_width          1.476 (MET)
...
```

To have the summary report show the violating endpoint, use the **-path_type end** option.

You can restrict the scope of the report by specifying one or more constraint-type options:

```
Command Option          Constraint Reported

-max_delay              max_delay & setup
-min_delay              min_delay & hold
-max_capacitance        max_capacitance
-min_capacitance        min_capacitance
-max_transition         max_transition
-min_transition         min_transition
-min_pulse_width        min_pulse_width
-recovery               asynchronous recovery
-removal                asynchronous removal
```

```
-max_skew               max_skew
-clock_gating_setup     clock gating setup
-clock_gating_hold      clock gating hold
-data_setup             data-to-data setup
-data_hold              data-to-data hold
-clock_setup            clock-to-clock setup
-clock_hold             clock-to-clock hold
-non_transparent_setup  nontransparent setup
-non_transparent_hold   nontransparent hold
```

To get a detailed report on what is causing the worst violation or what is closest to causing a violation, use the **-verbose** option. For example, the following two commands generate the same detailed path timing report:

```
nt_shell> report_constraint -max_delay -verbose

nt_shell> report_paths -max -max_paths 1 -path_type full
```

To get a report on all violating paths rather than just the worst or least-slack path, use the **-all_violators** option.

COMMAND PHASING

The **report_constraint** command can only be executed after path tracing is complete.

EXAMPLES

The following command generates a summary report that lists the constraints that have been evaluated, together with the amount of the worst violation or the smallest amount by which the constraint was met:

```
nt_shell> report_constraint
 ...
Constraint                    Slack
------------------------------ -------- --------
max_delay/setup                -0.653 (VIOLATED)
min_delay/hold                  0.102 (MET)
clock_tree_pulse_width          1.476 (MET)
 ...
```

The following command generates a report listing all of the worst setup timing violators:

```
nt_shell> report_constraint -max_delay -all_violators
 ...
max_delay/setup
                      Required   Actual
Pin                    Arrival  Arrival   Slack
--------------------- --------  ------- ------- ----------
Xsreg.Xreg54.X3.Mp0.g    3.045    3.698  -0.653 (VIOLATED)
Xsreg.Xreg56.X3.Mp0.g    3.045    3.698  -0.653 (VIOLATED)
Xsreg.Xreg62.X3.Mp0.g    3.045    3.692  -0.647 (VIOLATED)
Xsreg.Xreg53.X3.Mp0.g    3.045    3.679  -0.634 (VIOLATED)
Xsreg.Xreg57.X3.Mp0.g    3.045    3.679  -0.634 (VIOLATED)
Xsreg.Xreg61.X3.Mp0.g    3.045    3.677  -0.632 (VIOLATED)
```

```
Xsreg.Xreg52.X3.Mp0.g    3.045    3.669  -0.625 (VIOLATED)
 ...                       ...      ...     ...
Xsreg.Xreg36.X3.Mp0.g    3.045    3.045  -0.000 (VIOLATED)
```

The following command generates a detailed path report for the worst setup violation:

```
nt_shell> report_constraint -max_delay -verbose
 ...
max_delay/setup


Startpoint:  clk2 (in port)
Endpoint:    Xsreg.Xreg54.X3.Mp0.g
Path Type:   max
Constraint:  latch setup

Point                    NT     Incr   Adjust     Path
------------------------ -- -------- -------- -------- --
clk2 (in)                C     0.750             0.750  r
Xi0.X0.Mn0.g (inv2)            0.000             0.750  r
Xi0.X1.Mp0.g (inv2)            0.028             0.778  f
Xb00.X0.Mn0.g (inv2)           0.044             0.822  r

 ...

data arrival time                                3.698

Xsreg.Xreg54.Mn0.g (dff)       3.045             3.045
setup time                     0.000             3.045
data required time                               3.045
-----------------------------------------------------------
data required time                               3.045
data arrival time                               -3.698
-----------------------------------------------------------
slack (VIOLATED)                                -0.653
```

SEE ALSO

```
mark_clock_network(2)
report_paths(2)
set_non_transparent(2)
create_timing_check(2)
trace_paths(2)
```

report_crosstalk_delay_sources

Reports the aggressor nets and associated coupling capacitances of victim nets in crosstalk delay analysis.

SYNTAX

```
status report_crosstalk_delay_sources
   [-cost_type net_cost_type]
   [-max_nets N]
   [-max_aggressors N]
   [-min]
   [-max]
   [-min_active_aggressors N]
   [-delta_delay_ratio]
   [-aggressor_contributions]
   [-related_nets]
   [-significant_digits digits]
   [-nosplit]
   [nets]
```

Data Types

net_cost_type	string
N	integer
digits	integer
nets	list

ARGUMENTS

-cost_type *net_cost_type*

The cost type used for sorting the reported nets, either **delta_delay** or **delta_delay_ratio**. A value of **delta_delay** causes the nets to be reported in order of decreasing absolute delta delay, whereas a value of **delta_delay_ratio** causes the nets to be reported in order of decreasing relative delta delay (absolute delta delay divided by stage delay).

-max_nets *N*

The maximum number of victim nets to be reported. The default is 10.

-max_aggressors *N*

The number of aggressors reported for each victim net. The default is 1.

-min

Reports data used during minimum-delay analysis only.

-max

Reports data used during maximum-delay analysis only.

-min_active_aggressors *N*

The minimum number of active aggressors for a net to be reported. The default is 0, which means a net is reported if affected by any number of aggressors.

-delta_delay_ratio

Report the delta delay ratio for each victim net in addition to the delta delay.

-aggressor_contributions

Report estimates of the percentage contributions from each aggressor.

-related_nets

Reports the nets related to each net affected by crosstalk. If you use this option, a column is added to the report showing the nets that are related to each listed victim net. A net is related to a victim net if it is connected to that net through one or more source-drain channels of conducting transistors. Related nets reported in the table show the same delta delay numbers (Max Delta Delay, Max Ratio, Min Delta Delay, Min Ratio).

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

nets

A list of nets to report. If no list is specified, all nets with delta delay are reported.

DESCRIPTION

This command reports the aggressors and their associated coupling capacitances for victim nets that have crosstalk delta delay according to the worst delta delays or delta delay ratios. It can be used only after the **trace_paths** command has been run, and only when crosstalk analysis is enabled. You must have a NanoTime Ultra license to run crosstalk (signal integrity) analysis.

If coupling capacitance variation has been enabled, the column which displays an aggressor's coupling capacitance displays a range in the format **[min_cap, max_cap]**. The column which displays an

aggressor's percentage of the total coupling capacitance is based on nominal coupling capacitance values.

The cost function set with the **-cost_type** option determines the order in which the nets are reported:

delta_delay

The cost factor is calculated by determining the worst delta delay on the victim net. This is useful for finding the largest delta delays in the design that contribute to failing paths.

delta_delay_ratio

The cost factor is calculated by determining the worst delta delay ratio (ratio of delta delay to total stage delay) on the victim net. This cost factor is useful for finding the delta delays that result in the largest percentage change of stage delay in a failing path.

The reports for minimum-delay analysis and maximum-delay analysis might be different. By default, both are reported. Use the **-max** or **-min** option to restrict the scope of reporting.

To report only the victim nets that have more than a certain number of active aggressors, use the **-min_active_aggressors** option. The number of active aggressors is defined as the number of effective aggressors minus the number of completely quiet aggressors for a given situation. If neither the **-min** nor the **-max** option is specified, a completely quiet aggressor must be quiet for all four situations: min rise, min fall, max rise, and max fall. For maximum-delay reporting, a completely quiet aggressor must be quiet for max rise and max fall.

To report estimates of the percentage contributions to the total delay from each aggressor, use the **-aggressor_contributions** option.

COMMAND PHASING

The **report_crosstalk_delay_sources** command can be executed at any time after the **trace_paths** command.

EXAMPLES

The following command reports nets that have crosstalk delta delay for minimum-delay analysis, listed in order of decreasing delta delay, for the five nets with the largest delta delays, reporting the three aggressors with greatest coupling capacitance for each victim.

```
nt_shell> report_crosstalk_delay_sources -max_aggressors 3 \
          -max_nets 5 -min
 ...
                      rising   falling
coupling   percent aggressor aggressor min delta
     cap  of total     trans     trans     delay Victim net  Aggressor net
-------- --------- --------- --------- --------- ----------- -------------
  16.098     50.07     0.674     0.424    -0.295 x00.I2      x00.I1
  16.034     49.88     0.680     0.429    -0.295 x00.I2      x00.I3
   0.016      0.05     0.174     0.256    -0.295 x00.I2      Z2

  16.034     99.78     1.756     1.092    -0.013 x00.I3      x00.I2
   0.035      0.22     0.078     0.119    -0.013 x00.I3      Z3
```

```
        16.098     99.86     1.756     1.092    -0.013 x00.I1      x00.I2
         0.022      0.14     0.077     0.119    -0.013 x00.I1      Z1

         0.016    100.00     1.756     1.092    -0.003 Z2          x00.I2

         0.035    100.00     0.680     0.429    -0.002 Z3          x00.I3
```

SEE ALSO

report_paths(2)
si_enable_analysis(3)

report_delay_coefficients

Reports the delay coefficients specified for the delay arcs associated with the specified trigger or target objects.

SYNTAX

```
status report_delay_coefficients
   [-min]
   [-max]
   [-min_clock]
   [-max_clock]
   [-rise]
   [-fall]
   [-target target_objects]
   [trigger_objects]
```

Data Types

```
   target_objects          list
   trigger_objects         list
```

ARGUMENTS

-min

Reports delay coefficients for short data path analysis.

-max

Reports delay coefficients for long data path analysis.

-min_clock

Reports delay coefficients for short clock path analysis.

-max_clock

Reports delay coefficients for long clock path analysis.

-rise

Reports delay coefficients for rising slope.

-fall

Reports delay coefficients for falling slope.

-target ***target_objects***

The list of nets, transistor gate pins, or output ports reported by the command. If target objects are specified without any trigger objects, all coefficients to the target are reported. If both trigger objects and target objects are specified in the same command, the coefficients for the trigger-target pair is reported.

trigger_objects

The list of trigger nets or transistor gate pins reported by the command. This list defines a set of delay calculation trigger pins or nets.

DESCRIPTION

The **report_delay_coefficients** command reports coefficients that affect the calculated delay values at specified locations in the design.

In the command, you must specify the types of timing arcs which will be reported by the command (min data, max data, min clock, or max clock) and a list of objects that define the delay arcs which will be reported. You can specify the reported delay arcs with a list of trigger objects as an argument to the command, a list of target objects as an argument to the **-target** option, or both.

COMMAND PHASING

The **report_delay_coefficients** command can be executed any time after the **link_design** command.

SEE ALSO

set_delay_coefficients(2)
remove_delay_coefficients(2)

report_design

Displays attributes of the current design.

SYNTAX

```
report_design
   [-nosplit]
```

ARGUMENTS

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

DESCRIPTION

Generates a report about the attributes of the current design, including unit sizes and technology.

COMMAND PHASING

The **report_design** command can be executed at any time after the **link_design** command.

EXAMPLES

The following is an example of a design report.

```
nt_shell> current_design
```

```
{"ALU"}
nt_shell> report_design

****************************************
Report : design
Design : ALU
...
****************************************

  Design Attribute                    Value
-----------------------------------------------------
Units:
  Time Unit                           1 ns
  Capacitance Unit                    0.001 pF
  Voltage Unit                        1 V
  Resistance Unit                     0.001 kohm
  Current Unit                        0.001 A
  Width/Length Unit                   1 um
  Area Unit                           1 um square

Technology:
  max_technology                      typ
  min_technology                      typ
  max_clock_technology                typ
  min_clock_technology                typ
```

SEE ALSO

current_design(2)
report_hierarchy(2)
report_lib(2)

report_disable_timing

Reports disabled library cell arcs in the current design.

SYNTAX

```
status report_disable_timing
   [-nosplit]
   [cells]
```

Data Types

```
   cells          collection
```

ARGUMENTS

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

cells

Limits disabled arc reporting to the specified list of netlist cells. Provide the list or collection of cells as an argument to the command.

DESCRIPTION

Reports disabled library cell arcs in the current design.

Library cell arcs can be disabled in several ways. The C flag denotes the presence of conditional arcs (arcs that have a when statement defined in the library). The d flag denotes the presence of default arcs.

EXAMPLES

The following report shows that the library cell arc of cell *Xmux2* from pin *S* to pin *Z* are disabled as a result of a case-analysis constant of *0* set on input pin *I0*.

```
nt_shell> set_case_analysis 0 {in0}
nt_shell> report_disable_timing

****************************************
Report : disable_timing
Design : middle
****************************************

Flags :     C  Conditional arc
            d  default arc

Cell or Port         From    To      Sense           Flag  Reason
-------------------------------------------------------------------------
Xmux2                S       Z       negative_unate  C     I0&!I1 = 0
Xmux2                S       Z       negative_unate  d

1
```

SEE ALSO

```
set_case_analysis(2)
report_lib(2)
```

report_exceptions

Generates a report of timing exceptions.

SYNTAX

```
status report_exceptions
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-nosplit]
```

Data Types

```
   from_list               list
   rise_from_list          list
   fall_from_list          list
   through_list            list
   rise_through_list       list
   fall_through_list       list
   to_list                 list
   rise_to_list            list
   fall_to_list            list
```

ARGUMENTS

-from *from_list*

Specifies a list of pins, ports, nets, or clocks considered path startpoints, for restricting the scope of the report. Only exceptions that were defined with a matching **-from** option are reported.

-rise_from *rise_from_list*

Reports only exceptions that were defined with a matching **-rise_from** option.

-fall_from *fall_from_list*

Reports only exceptions that were defined with a matching **-fall_from** option.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass to be included in the report. Only exceptions that were defined with either a matching **-through**, or **-exclude_reference_through** option (in the case of exclude_reference exceptions) are reported.

-rise_through ***rise_through_list***

Reports only exceptions that were defined with a matching **-rise_through**, **-rise_common_through**, or **-exclude_reference_rise_through** option.

-fall_through ***fall_through_list***

Reports only exceptions that were defined with a matching **-fall_through**, **-fall_common_through**, or **-exclude_reference_fall_through** option.

-to ***to_list***

Specifies a list of pins, ports, nets, or clocks to be considered path endpoints, for restricting the scope of the report. Only exceptions that were defined with a matching **-to** option are reported.

-rise_to ***rise_to_list***

Reports only exceptions that were defined with a matching **-rise_to** option.

-fall_to ***fall_to_list***

Reports only exceptions that were defined with a matching **-fall_to** option.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line splitting, resulting in long, unbroken lines in the report.

DESCRIPTION

The **report_exceptions** command reports the timing exceptions previously defined by the following commands:

```
set_exclude_reference_path
set_false_path
set_no_timing_check_path
set_multicycle_path
set_no_same_phase_path
set_same_phase_path
```

For each exception defined by one of these commands, the report lists the objects specified as the "from," "through," and "to" points; the type of exception applied for setup and hold checks; and the number of times the exception was matched during path tracing.

The **report_exceptions** command with no options reports all exceptions. To restrict the scope of the report, use the same the "from," "through," and "to" options that were used in the original exception-

setting commands.

A general path specification reports a matching general or more-specific exception if there is a matching object in the path specification of the original exception-setting command. For example, the **report_exceptions -from selA** command reports all exceptions originally specified with the **-from selA**, **-rise_from selA**, and **-fall_from selA** options.

To cancel an exception listed in the report, use the remove_* command corresponding to the original exception-setting command:

remove_exclude_reference_path
remove_false_path
remove_no_timing_check_path
remove_multicycle_path
remove_no_same_phase_path
remove_same_phase_path

To report path-tracing restrictions set with the **set_find_path** command, use the **report_find_path** command.

COMMAND PHASING

The **report_exceptions** command can be executed any time after the **link_design** command.

EXAMPLES

The following example lists all timing exceptions set on the design:

```
nt_shell> report_exceptions
 ...

From     Through   To          Setup          Hold          Matches
-------- --------- ---------- -------------- ------------ -------
selA     *         BUS[8]     FALSE          FALSE              4
selA(r)  *         {and add}  FALSE          FALSE              4
*        {m1 m2}   *          cycle=3        --                28
cyc1     *         *          no_same_phase  no_same_phase      8
cyc0     *         *          same_phase     same_phase        16
test     *         *          no_check       no_check         168
*        {m1}
         {m2(e)}   *          --             exclude_ref        2
```

The From, Through, and To columns show the objects specified by the **-from**, **-through**, and **-to** options used in the original exception-setting command. The notation (r) or (f) after the object name indicates rising or falling edges. The notation (e) after the object name in the Through column indicates an exclude_reference_through pin or net as specified in the original exclude_reference exception command.

The Setup and Hold columns indicate the type of exceptions applied for setup and hold checking: common_req, exclude_ref, false path, no_same_phase checking, same_phase checking, no_check status, or multicycle checking.

Multicycle checks are displayed with the formats "cycle=3" and "abs_cycle=3." In the Setup column,

entries are always in the format "cycle=3" and they result from **set_multicycle_path -setup** commands. In the Hold column, entries can be in either format before path tracing. The format "cycle=3" results from **set_multicycle_path -hold** commands and the format "abs_cycle=3" results from **set_multicycle_path -hold_cycle** commands. After path tracing, all entries in the Hold column have the format "abs_cycle=3" because NanoTime converts all settings to a consistent format during path tracing.

The Matches column indicates the number of times the exception was matched during path tracing. Two dashes (--) indicate either that path tracing has not been done or that no matches occurred during path tracing. The number of matches reported for a particular exception-setting command such as the **set_multicycle_path** command can be affected by conflicting higher-priority exceptions, such as false path exceptions. The number of matches reported for exclude_reference exception commands count the number of launch path matches at timing check points when a capture path is excluded because it passed through the -exclude_reference_through point.

SEE ALSO

```
remove_exclude_reference_path(2)
remove_false_path(2)
remove_find_path(2)
remove_multicycle_path(2)
remove_no_same_phase_path(2)
remove_no_timing_check_path(2)
remove_same_phase_path(2)
report_find_path(2)
set_exclude_reference_path(2)
set_false_path(2)
set_find_path(2)
set_multicycle_path(2)
set_no_same_phase_path(2)
set_no_timing_check_path(2)
set_same_phase_path(2)
```

report_fanin

Reports all pins or instances that cause timing check exceptions in the transitive fanin cone of target pins.

SYNTAX

```
status report_fanin
   [-false_path]
   [-no_timing_check_path]
   [-constant_origin]
   [-dont_search_thru]
   [-previous_check]
   [-all]
   target_pins
```

Data Types

```
   target_pins   list
```

ARGUMENTS

-false_path

Report all pins in the transitive fanin cone of target pins that are the last pin triggering a user-specified false path (a path specified by a **set_false_path** command).

-no_timing_check_path

Report all pins in the transitive fanin cone of target pins that are the last pin triggering a user-specified no-check path (a path specified by a **set_no_timing_check_path** command).

-constant_origin

Report all pins in the transitive fanin cone of target pins) that are origins of fixed logic.

-dont_search_thru

Report all instances in the transitive fanin cone of target pins that are marked with the **mark_instance -dont_search_thru_gate** or **mark_instance -dont_search_thru_channel** command.

-previous_check

Report all pins in the transitive fanin cone of target pins that have user-specified timing checks created

without using the **-continue** option of the **create_timing_check** or **set_timing_check_attributes** command.

-all

Generate one report for all available options.

target_pins

A list of pins for which to report all pins or instances that trigger timing check exceptions in the transitive fanin cone.

DESCRIPTION

The **report_fanin** command generates a report of all pins or instances that trigger timing check exceptions in the transitive fanin cone of user-specified target pins. Reported pins and instances are listed in alphanumeric order.

You must specify at least one of the **-false_path**, **-no_timing_check_path**, **-constant_origin**, **-dont_search_thru**, **-previous_check**, and **-all** options. A separate report is built for each option used. In the case of the **-all** option, a separate report is created for all available options.

You must set the **timing_analysis_coverage_fanin** variable to **true** before running the **trace_paths** or **extract_model** command. This variable setting enables additional data collection and monitoring of timing checks related to fanin during path tracing. Note that setting the **timing_analysis_coverage_fanin** variable to **true** forces the **timing_analysis_coverage** variable to be **true**.

COMMAND PHASING

The **report_fanin** command produces useful results only after path tracing is complete.

EXAMPLES

The following example shows a fanin report for the **-false_path** option.

```
nt_shell > report_fanin -false_path XI3.XI2.MN1.g

target-pin           pins in transitive fanin cone
-------------------- ----------------------------------------
XI3.XI2.MN1.g        XI0.XI2.MP1.g,XI1.XI2.MP1.g
```

The following example shows two fanin reports for the **-false_path** and **-no_timing_check_path** options.

```
nt_shell> report_fanin -false_path -no_timing_check_path XI4.XI2.*.g

**************************************
```

```
Report : fanin coverage report for option false_path
Design :
Version:
Date   :
*************************************
target-pin           pins in transitive fanin cone
-------------------- -----------------------------------------
XI4.XI2.MN1.g        XI1.XI2.MN1.g
XI4.XI2.MP1.g        XI1.XI2.MP1.g

*************************************
Report : fanin coverage report for option no_timing_check_path
Design :
Version:
Date   :
*************************************
target-pin           pins in transitive fanin cone
-------------------- -----------------------------------------
XI4.XI2.MN1.g        XI5.MP1.g
XI4.XI2.MP1.g        XI5.MN1.g
```

SEE ALSO

trace_paths(2)
extract_model(2)
report_paths(2)
report_analysis_coverage(2)
timing_analysis_coverage(3)
timing_analysis_coverage_fanin(3)

report_fanout_noise

Reports one-level propagation of noise values onto the output nets of channel-connected regions driven by the victim.

SYNTAX

```
status report_fanout_noise
   [-nworst N]
   [-slack_lesser_than slack]
   [-height_greater_than height]
   [-significant_digits digits]
   [-nosplit]
   [-shape]
   [objects]
```

Data Types

```
N          integer
slack      float
height     float
digits     integer
objects    list
```

ARGUMENTS

-nworst *N*

The number of pins that will be have fanout noise values reported. By default, a single pin will have fanout values displayed.

-slack_lesser_than *slack*

Reports all fanout nets that have fanout noise slack values less than the specified value.

-height_greater_than *height*

Reports all fanout nets that have fanout noise peak values greater than the specified value.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the

number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

-shape

Reports the width and time-to-peak ratio of fanout noise bumps.

objects

A list of pins, or nets from which fanout values will be reported. If no list is specified, all pins with noise values will be considered.

DESCRIPTION

This command reports the effect of propagating noise waveforms from pins that have noise through the driven channel-connected region to the fanout net. The original noise victim pin and the noise peaks within the rails are listed in the report. The fanout net and propagated noise values are also listed by type.

The command can be used only after the **update_noise** command has been executed, and only when the **si_enable_noise_analysis** and **si_enable_noise_fanout_analysis** variables are **true**.

Use the **-slack_lesser_than** option to select a set of fanout nets whose noise slack is less than the specified value.

Use the **-height_greater_than** option to select a set of fanout nets whose noise height is greater than the specified value.

COMMAND PHASING

The **report_fanout_noise** command can be executed at any time after the **trace_paths** command and the **update_noise** commands have been run.

EXAMPLES

The following command reports the pins that have the 3 worst noise slack values.

```
nt_shell> report_fanout_noise -nworst 3
 ...
  fanout    fanout Fanout             Trigger   Trigger
   slack    height Type   Fanout Net  Height AL Height BH Trigger Pin
-------- --------- ------ ----------- --------- --------- ----------------
-0.01927  -0.01927 BH     Z2          0.50531   -0.61284  x00.xbuf5.Mm_0.g
-0.05797   0.05797 AL     Z2          0.50531   -0.61284  x00.xbuf5.Mm_0.g
```

```
    -0.01927  -0.01927 BH     Z2          0.50531   -0.61284  x00.xbuf5.Mm_3.g
    -0.05797   0.05797 AL     Z2          0.50531   -0.61284  x00.xbuf5.Mm_3.g

    -0.09251  -0.09251 BH     Z4          0.47848   -0.59501  x00.xbuf7.Mm_0.g
    -0.01772   0.01772 AL     Z4          0.47848   -0.59501  x00.xbuf7.Mm_0.g
```

SEE ALSO

```
update_noise(2)
report_noise(2)
report_noise_violation_sources(2)
set_fanout_noise_threshold(2)
si_enable_noise_analysis(3)
si_enable_noise_fanout_analysis(3)
si_fanout_noise_margin_above_low(3)
si_fanout_noise_margin_below_high(3)
si_fanout_noise_threshold_above_low(3)
si_fanout_noise_threshold_below_high(3)
```

report_find_path

Reports the scope of paths previously selected by the **set_find_path** command.

SYNTAX

```
status report_find_path
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-nosplit]
```

Data Types

```
   from_list               list
   rise_from_list          list
   fall_from_list          list
   to_list                 list
   rise_to_list            list
   fall_to_list            list
   through_list            list
   rise_through_list       list
   fall_through_list       list
```

ARGUMENTS

-from *from_list*

Specifies a list of pins, ports, nets, or clocks considered path startpoints. If you specify a clock, all potential startpoints clocked by that clock are considered.

-rise_from *rise_from_list*

Same as the **-from** option, except that only rising-edge transitions from the specified objects are considered.

-fall_from *fall_from_list*

Similar to the **-rise_from** option, except for falling-edge transitions.

-to ***to_list***

Specifies a list of pins, ports, nets, or clocks to be considered path endpoints, If you specify a clock, all potential endpoints clocked by that clock are considered.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_to ***fall_to_list***

Similar to the **-rise_to** option, for falling-edge transitions.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass to be considered.

-rise_through ***rise_through_list***

Same as the **-through** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_through ***fall_through_list***

Same as the **-through** option, except that only falling-edge transitions arriving at the specified objects are considered.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

DESCRIPTION

The **report_find_path** command reports the scope of paths previously selected by the **set_find_path** command. The **report_find_path** command by itself, without any options, reports all paths currently selected by previous **set_find_path** commands. You can optionally use the **-from**, **-to**, **-through**, and similar options to restrict the scope of the report.

To remove paths from the selection set, use the **remove_find_path** command.

COMMAND PHASING

The **report_find_path** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports the current scope of all paths previously selected with the **set_find_path** command.

```
nt_shell> report_find_path
```

The following example shows how to restrict the scope of the report with the **-from** option.

```
nt_shell> set_find_path \
          -from [get_ports selA] \
          -to [get_ports BUS[60]]
1
nt_shell> set_find_path \
          -from [get_ports selB] \
          -to [get_ports BUS[61]]
1
nt_shell> report_find_path


From            Through         To
--------------- --------------- --------
selA            *               BUS[60]
selB            *               BUS[61]

1
nt_shell> report_find_path -from [get_ports selA]
...
From            Through         To
--------------- --------------- --------
selA            *               BUS[60]
```

SEE ALSO

```
check_design(2)
set_find_path(2)
remove_find_path(2)
trace_paths(2)
```

report_hierarchy

Reports the reference hierarchy of the current instance or current design.

SYNTAX

```
status report_hierarchy
   [-full]
   [-noleaf]
   [-nosplit]
   [-maxdepth depth]
```

Data Types

```
   depth       integer
```

ARGUMENTS

-full

Displays the full hierarchy of every instance, rather than just the first instance, when there are multiple instances of the same block.

-noleaf

Suppresses the display the leaf library cells.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

-maxdepth depth

Limits the hierarchical depth of the report to a specified number, an integer of at least 1.

DESCRIPTION

The **report_hierarchy** command lists the hierarchy of the design, block by block, down to the lowest levels of hierarchy. Blocks within a block are shown indented below the name of the parent block. The list shows the reference name and, if it is a library block, the library name. It does not show instance names. To view a list of instance names, use the **report_cell** command.

If the current instance has been set with the **current_instance** command, the report shows the hierarchy of that instance. Otherwise, the report shows the hierarchy of the current design.

To limit the hierarchical depth of the report (and therefore the number of levels of indenting), use the **-maxdepth** option, specifying a value of 1 or more. The smaller the number, the shorter the report.

By default, if multiple instances of a block exist, the contents of the block are shown only for the first instance in the report. Subsequent times, the report shows an ellipsis (...). To show the full hierarchy for all instances in the design, use the **-full** option.

To suppress the display of leaf-level library cells, use the **-no_leaf** option.

COMMAND PHASING

The **report_hierarchy** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports the hierarchy of the design.

```
nt_shell> report_hierarchy
...
Instance                      Library
---------------------------------------------
  reg64
    dff
      nand2
        transistor            builtin_elements
        transistor            builtin_elements
        transistor            builtin_elements
        transistor            builtin_elements
      inv2
        transistor            builtin_elements
        transistor            builtin_elements
...
```

The following command reports the hierarchy of the design, but limits the depth of the report to 2 levels.

```
nt_shell> report_hierarchy -maxdepth 2
...
```

```
Instance                      Library
-----------------------------------------------
  reg64
    dff
    dff
      ...
    dff
      ...
...
```

SEE ALSO

current_design(2)
current_instance(2)
report_cell(2)

report_lib

Reports technology library information.

SYNTAX

```
status report_lib
   [-timing_arcs]
   [-nosplit]
   library_name
   [lib_cell_list]
```

Data Types

```
   library_name        list
   lib_cell_list       list
```

ARGUMENTS

-timing_arcs

Displays information on all cell timing arcs.

-nosplit

Prevents the splitting of lines in the report, resulting in long, unbroken lines when information in a given field exceeds the column width.

library_name

A pattern that matches a single library or a collection containing one library. The library must have been read into NanoTime in .db format with the **read_library** command.

lib_cell_list

A list of library cells about which information is reported. The default is to report information about all cells in the library. Each element in this list is either a collection of library cells or a pattern that matches library cells in the library.

DESCRIPTION

The **report_lib** command displays a report on a specified library, showing information about units, operating conditions, and available library cells. The **-timing_arcs** option adds detailed timing arc information for each library cell.

To generate a report, the specified library must be loaded into NanoTime. To find out which libraries are loaded and available, use the **list_libraries** command.

COMMAND PHASING

The **report_lib** command is not phase-restricted.

EXAMPLES

The following example lists the available libraries and reports on the single library currently being used.

```
nt_shell> list_libs

Library Registry:
    builtin_elements   builtin_elements:None

1
nt_shell> report_lib *
**************************************
Report : library
Design : builtin_elements
...
**************************************

Time Unit                      : 1 ns
Capacitance Unit               : 0.001 pF
Voltage Unit                   : 1 V
Resistance Unit                : 0.001 kohm
Current Unit                   : 1 A

Operating Conditions:

  Name        Process    Temp   Voltage  Tree Type
  -------------------------------------------------
  typ         1.00       25.00  2.00     balanced_case

Library Cells:

  Attributes:

   b   -  black box (function unknown)
   d   -  dont_touch
   s   -  state table
   u   -  dont_use
   A   -  abstracted timing model
   E   -  extracted timing model
   I   -  Interface timing spec model (ITS)
```

```
      S   -  Stamp timing model
      Q   -  Quick timing model (QTM)

   Lib Cell        Attributes
   -------------------------
   capacitor       --
   grounded_cap    --
   resistor        --
   transistor      --

   1
```

SEE ALSO

```
list_libs(2)
read_library(2)
```

report_logic_constraint

Reports logic constraints previously set with the **set_logic_constraint** command.

SYNTAX

```
status report_logic_constraint
   [-print_logic_state]
   [-related]
   [-conflicts]
   [-hierarchical]
   [-constraints constraint_list]
   [pins_ports_or_nets]
```

Data Types

```
   constraint_list              list
   pins_ports_or_nets           list
```

ARGUMENTS

-print_logic_state

Reports the fixed logic states of the nets, if any, resulting from the **set_case_analysis** command.

-related

Reports all constraints that are logically related to the specified pins, ports, or nets by chains of constraints, in addition to the constraints on the immediately affected nets.

-conflicts

Reports the nets that are not in a fixed logic state and cannot be set to logic high or logic low due to multiple constraints (typically a set of constraints with circular dependency that cannot be resolved).

-hierarchical

Reports constraints set at lower levels of the hierarchy as well as at the top level of the current design.

-constraints *constraint_list*

Restricts reporting to the specified list of constraint types. Valid values are invert, one_off, one_hot, at_most_one_off, at_most_one_hot, equal, nand, nor, and disable.

pins_ports_or_nets

The list of ports, pins, or nets for which to report logical constraints.

DESCRIPTION

The **report_logic_constraint** command reports logic constraints previously set on ports, pins, or nets with the **set_logic_constraint** command.

You can optionally specify the same list of pins or nets as the original **set_logic_constraint** command to get a report on just that set of pins or nets. Otherwise, NanoTime reports all logic constraints that have been set on the current design.

To report the constraints affecting nets in the fanin and fanout of the specified objects, use the **-related** option. For example, in a long inverter chain, if you specify one net in the chain, by default the command reports only the logical constraints on that net and the two adjacent nets. With the **-related** option, all nets in the inverter chain are reported.

To restrict the types of logic constraints that are reported, use the **-constraints** option and list the constraint types: invert, one_off, one_hot, at_most_one_off, at_most_one_hot, equal, or disable. In the absence of this option, all logic constraints are reported.

COMMAND PHASING

The **report_logic_constraint** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports the logic constraint previously set on signals ct0 through ct3.

```
nt_shell> report_logic_constraint \
          [get_nets {ct0 ct1 ct2 ct3}]
 ...
Constraint      Nets
-----------------------------------------------
one_hot         {ct0 ct1 ct2 ct3}
```

The following command reports all invert and equal logic constraints that have been set on the current design.

```
nt_shell> report_logic_constraint \
          -constraints {invert equal}
```

SEE ALSO

remove_logic_constraint(2)
set_case_analysis(2)
set_logic_constraint(2)

report_logic_state

Reports the logic state of pins, ports, or nets resulting from case analysis or logic constraints.

SYNTAX

```
status report_logic_state
   [-nosplit]
   [-valid_only]
   [pins_ports_or_nets]
```

Data Types

```
   pins_ports_or_nets      list
```

ARGUMENTS

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

-valid_only

Reports only the nets that have a logic state of 0 or 1; skips reporting of nets that have a logic state of X.

pins_ports_or_nets

A list of pins, ports, or nets to report. If you do not specify list, the command reports all nets.

DESCRIPTION

The **report_logic_state** command reports the logic states of nets in the design. If you specify a list of pins, ports, or nets, the command reports the logic state of each net associated with the listed objects: 1,

0, or X (where X represents "unknown"). If you do not specify a list of objects, the command reports all nets. If you use the **-valid_only** option, the command reports only the nets that have a logic state of 0 or 1; it does not report the unknown nets.

The following is an example of a report:

```
Logic State Attribute:
    u - User set
    i - Implied

Nets            State  Attributes
--------------- ------ ----------
BUS[60]         1      u
BUS[61]         0      u
add             0      u
addsub          0      i
addsub0         0      i
...
```

Logic states can be set on ports or pins with the **set_case_analysis** command and on pins or nets with the **set_logic_constraint** command. In addition, logic states that have been set by these commands can be propagated to other nets through the transistors or the combinational cell functions. The "Attributes" column of the report indicates whether the logic state was set directly by the user ("u") or determined implicitly by propagation ("i").

COMMAND PHASING

The **report_logic_state** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports all nets that have a logic state of either 0 or 1.

```
nt_shell> report_logic_state -valid_only
```

The following command reports the logic state (0, 1, or X) of each net that begins with the letter "S".

```
nt_shell> report_logic_state [get_nets S*]
```

SEE ALSO

```
set_case_analysis(2)
set_logic_constraint(2)
```

report_lvs_equivalent_nets

Reports the nets that have been declared to be equivalent.

SYNTAX

```
status report_lvs_equivalent_nets
```

DESCRIPTION

The **report_lvs_equivalent_nets** command reports all nets that have been declared to be equivalent with the **set_lvs_equivalent_nets** command.

COMMAND PHASING

The **report_lvs_equivalent_nets** command is used after the **link_design** command.

EXAMPLES

The following command reports the net equivalences after nets n1 and n2 have been declared to be equivalent nets.

```
nt_shell> report_lvs_equivalent_nets
 ...
Master Net      Synonyms
--------------- ----------------------------------------------------------
n1              n2
```

SEE ALSO

```
set_lvs_equivalent_nets(2)
```

report_measurement

Reports the differential voltage across sense amplifier inputs when the sense amplifier enable signal crosses a threshold.

SYNTAX

```
status report_measurement
   [-name measurement_name]
   [-violators]
   [-trigger net]
   [-nosplit]
```

Data Types

```
   measurement_name      string
   net                   string
```

ARGUMENTS

-name *measurement_name*

Displays only the measurements with the specified names. Wildcard use is supported.

-violators

Displays only the measurements that result in a violation.

-trigger *net*

Displays measurements which are taken in the context of the delay arc whose trigger net is listed as the argument to this option.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

DESCRIPTION

The **report_measurement** command evaluates the differential voltage across the sense amplifier inputs when the sense amplifier enable signal crosses a user-defined threshold. The differential voltage is compared to a user-defined check value and is considered to be a violation if it is below that value.

In this analysis, NanoTime considers the voltages at every pin of each of the two sense amplifier input nets. For example, assume that one of the input nets includes pins A1, A2, and A3. The other input net includes pins B1, B2, and B3. NanoTime first calculates the differences between the voltage at pin A1 and the voltages at pins B1, B2, and B3. The tool repeats the calculation for pins A2 and A3, resulting in a total of nine calculated voltage difference values.

NanoTime reports the smallest and largest values of the calculated voltage differences. The smallest value is labeled "min" under the column header "Keep value type", while the largest value is labeled "max".

The report contains up to four lines, with a minimum and maximum voltage difference for each of the minimum-delay and maximum-delay paths. However, if the minimum and maximum voltage differences are equal, only the maximum value is reported.

To specify the reporting threshold, set the **timing_memory_measurement_sense_amp_trigger_threshold_percentage** variable. To specify the percentage of the supply voltage below which a measurement is considered to be a violation, set the **timing_memory_measurement_differential_voltage_percentage** variable.

If you use the **report_measurement** command without any options, the report displays all measurements. The report includes the measurement name, the measurement result, the trigger net, the trigger direction, the search path type, the check value type and the check value. The report also includes the switching net and the measured nets for differential voltage measurements.

You can display a subset of the available measurements by using the **-name** option to list the measurements matching the specified names. The option supports wildcard characters.

Use the **-violators** option to list the measurements which resulted in a violation.

The **-trigger** option applies to the trigger net of the delay arc which is in the context where the measurement is taken. Limit the report by specifying the trigger net of the delay arc.

COMMAND PHASING

The **report_measurement** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports measurements with trigger net "Xmem/rwaddr0."

```
nt_shell> report_measurement -trigger Xmem/rwaddr0

Switching Direction:
    f - Falling
    r - Rising
```

```
                                        Keep
         Diff                   Path  value   Check
Name     voltage Trigger        type   type   value Switch net Measured nets
-------- ------- ------------ -- ----  -----  ------ ---------- --------------------
diffV_0  0.1000 Xmem/rwaddr0 r   max    min  0.1200 Xmem/saen  {Xmem/data0 Xmem/datab0}
diffV_1  0.2000 Xmem/rwaddr0 r   max    max  0.1200 Xmem/saen  {Xmem/data0 Xmem/datab0}

1
```

The following command reports the measurement with the name diffV_1.

```
nt_shell> report_measurement -name diffV_1

Switching Direction:
    f - Falling
    r - Rising


                                        Keep
         Diff                   Path  value   Check
Name     voltage Trigger        type   type   value Switch net Measured nets
-------- ------- ------------ -- ----  -----  ------ ---------- --------------------
diffV_1  0.2000 Xmem/rwaddr0 r   max    max  0.1200 Xmem/saen  {Xmem/data0 Xmem/datab0}

1
```

SEE ALSO

```
mark_sense_amp(2)
erase_sense_amp(2)
report_topology(2)
timing_memory_measurement_sense_amp_trigger_threshold_percentage(3)
timing_memory_measurement_differential_voltage_percentage(3)
```

report_memory

Generates a report on the memory in the design.

SYNTAX

```
status report_memory
   [-wordlines]
   [-bitlines]
   [-sense_amps]
   [-nosplit]
   [-significant_digits digits]
   [memory_list]
```

Data Types

```
   digits          integer
   memory_list     list
```

ARGUMENTS

-wordlines

Reports the nets identified as wordlines.

-bitlines

Reports the nets identified as bitlines.

-sense_amps

Reports the sense amplifiers in the memory.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the

report_default_significant_digits variable, whose default is 3.

memory_list

A list of memories to be reported. Each element in this list is either a collection of memories or a pattern matching the memory names. Without this option, all memories are reported.

DESCRIPTION

This command displays information about memories in the current design. The types of memory information reported by the command depend on the options used. By default, the command produces a brief report on all memories in the design. For each named memory, the report includes the number of wordlines, number of bitlines, number of bit cells, the analysis mode set by the **create_memory** command, the analysis type, the number of bit cells selected, and the top hierarchical cell. When the *memory_list* option is specified, only the selected memories are reported.

Use the **-wordlines** option to generate an extra report section summarizing the net name for each associated wordline. Use the **-bitlines** option to generate an extra report section summarizing the bitline net name and its complement for each associated bitline. Use the **-sense_amps** option to generate an extra report section summarizing the enable pin, input net names, and output net names of each associated sense amp.

COMMAND PHASING

The **report_memory** command can be executed at any time after the **link_design** command.

EXAMPLES

This example shows the default memory report.

```
nt_shell> report_memory
  ...
           # Word    # Bit    # Bit Analysis Analysis   # Bits
Name        Lines    Lines    Cells Mode     Type     Selected Top Cell
--------- ------ -------- -------- -------- -------- -------- --------
read_mem        2        4        4 read     minmax(1)        4 --
```

SEE ALSO

```
create_memory(2)
get_memory(2)
```

report_merged_nets

Generates a report on the nets that have been merged with the **set_merged_nets** command.

SYNTAX

```
status report_merged_nets
   net_list
```

Data Types

```
   net_list          list
```

ARGUMENTS

net_list

One or more nets that have been merged.

DESCRIPTION

This command generates a report on the nets that have been merged previously with the **set_merged_nets** command. Specify at least one net in a set of merged nets in the command argument. The report shows all the nets belonging to that merged set.

COMMAND PHASING

The **report_merged_nets** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports all nets that have been previously merged with net c3 by the **set_merged_nets** command.

```
nt_shell> report_merged_nets c3
```

SEE ALSO

remove_merged_nets(2)
set_merged_nets(2)

report_min_pulse_width

Reports minimum pulse width violations set with the **set_min_pulse_width** command and detected during path tracing.

SYNTAX

```
status report_min_pulse_width
   [-nets]
   [-verbose]
   [-nosplit]
   [-significant_digits digits]
   [port_pin_list]
```

Data Types

```
   digits          integer
   port_pin_list   list
```

ARGUMENTS

-nets

Includes the net names.

-verbose

Reports the details of the pulse width calculation, including path arrival times and delays, rather than just the pin name and pulse width.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

port_pin_list

The list of objects to check.

DESCRIPTION

The **report_min_pulse_width** command reports minimum pulse width violations found during path tracing. Pulse width constraints are set with the **set_min_pulse_width** command before the **check_design** command. Reporting of violations can only be done after the **trace_paths** command.

By default, the **report_min_pulse_width** command lists the pins at which violations were detected and shows the required minimum pulse width, actual pulse width, and slack. To get a detailed report showing the path arrival times and delays used in the pulse width calculation, use the **-verbose** option. To restrict the scope of the report to specific ports or pins, specify a list of the objects of interest.

COMMAND PHASING

The **report_min_pulse_width** command can be executed at any time after the **trace_paths** command.

EXAMPLES

The following command generates a list of all minimum pulse width violations:

```
nt_shell> report_min_pulse_width

...
                Required       Actual
Pin          Pulse Width  Pulse Width    Slack
------------ -----------  ----------- -------- ----------
Xlc8.Mn12.g        0.005       -0.004   -0.009 (VIOLATED)
Xlc9.Mn12.g        0.005       -0.004   -0.009 (VIOLATED)
Xlc10.Mn12.g       0.005       -0.004   -0.009 (VIOLATED)
Xlc11.Mn12.g       0.005       -0.004   -0.009 (VIOLATED)
Xlc12.Mn12.g       0.005       -0.004   -0.009 (VIOLATED)
...
```

The following command reports both high and low pulse width violations at pin Xlc8.Mn12.g:

```
nt_shell> report_min_pulse_width [get_pins Xlc8.Mn12.g]
...

                Required      Actual
Pin          Pulse Width Pulse Width    Slack
------------ ----------- ----------- -------- ----------
Xlc8.Mn12.g        0.005      -0.004   -0.009 (VIOLATED)
Xlc8.Mn12.g        0.006       0.004   -0.002 (VIOLATED)
```

The following command reports in detail the pulse width violations at pin Xlc8.Mn12.g:

```
nt_shell> report_min_pulse_width [get_pins Xlc8.Mn12.g] -verbose
...

Startpoint:  xor (in port)
Endpoint:    Xlc8.Mn12.g
Path Type:   min
Constraint:  user specified min pulsewidth check

Point                     NT     Incr   Adjust     Path
------------------------- -- -------- -------- -------- --
clock MCLK (rise)                0.000
input external delay             0.100
xor (in)                  D      0.000              0.100  f
Xxor1.X0.Mp0.g (inv2)            0.000              0.100  f
Xxor1.X1.Mn0.g (inv2)            0.034              0.134  r
Xlc8.Mn12.g (logic_cell)         0.025              0.159  f
open edge arrival time                              0.159

clock MCLK (rise)                0.000
input external delay             0.100
xor (in)                  D      0.000              0.100  r
Xxor1.X0.Mn0.g (inv2)            0.000              0.100  r
Xxor1.X1.Mp0.g (inv2)            0.028              0.128  f
Xlc8.Mn12.g (logic_cell)         0.027              0.155  r
close edge arrival time                             0.155
-------------------------------------------------------------
required pulse width (low)                          0.005
actual pulse width                                 -0.004
-------------------------------------------------------------
slack (VIOLATED)                                   -0.009
...
```

SEE ALSO

check_design(2)
report_constraint(2)
set_min_pulse_width(2)
trace_paths(2)

report_multi_input_switching

Reports the reselection and exclusion of nets during timing-based multi-input switching analysis.

SYNTAX

```
status report_multi_input_switching
   [-reselected]
   [-excluded]
```

ARGUMENTS

-reselected

Reports nets chosen for reselection during iterations of timing-based multi-input switching analysis.

-excluded

Reports nets excluded from reselection.

DESCRIPTION

This command reports the nets that have been previously chosen for reselection or excluded from reselection 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.

Use the **-reselected** or **-excluded** option to reduce the scope of the report. Otherwise, all reselected or excluded nets are reported.

COMMAND PHASING

The **report_multi_input_switching** command is not phase-restricted.

EXAMPLES

The following is an example of a multi-input switching report. Attribute E indicates that the net is excluded. Attribute R indicates that the net is reselected.

```
nt_shell report_multi_input_switching
 ...
 Output net         Attributes
 ------------------ -------------------------
 W23gat             R (min fall)
 W11gat             E
 W10gat             R (max rise)  (max fall) (min rise) (min fall)
```

Net W11gat (including its channel-connected region) is excluded for multi-input switching analysis. Nets W23gat and W10gat are reselected. The attributes also show for which edges (rise or fall) and for which paths (minimum delay or maximum delay) the reselection applies during the most recent analysis iteration. In this example, net W23gat is reselected only for the falling edge in minimum delay analysis.

SEE ALSO

exclude_multi_input_switching(2)
timing_enable_multi_input_switching(3)
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)

report_net

Generates a report of net information.

SYNTAX

```
status report_net
   [-connections]
   [-verbose]
   [-significant_digits digits]
   [-nosplit]
   [net_names]
```

Data Types

```
   digits              integer
   net_names           list
```

ARGUMENTS

-connections

Displays a list of pins connected to the net.

-verbose

Displays net capacitance information. This option can be used only with the **-connections** option.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable, whose default is 3.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

net_names

Specifies a list of nets to be reported. If no list is specified, the command reports all nets in the current

instance or current design.

DESCRIPTION

The command displays information about specified nets or all nets. If a current instance is set, the report is generated for that instance. Otherwise, the report is generated for the current design.

The **-connections** option lists the leaf-level pins connected to each net. The **-verbose** option shows net capacitance information.

COMMAND PHASING

The **report_net** command can be executed at any time after the **link_design** command.

EXAMPLES

The following example reports all nets in the design.

```
nt_shell> report_net

Attributes:
    s - supply
    g - part of global net

Net                Fanin   Fanout    Inout     Pins Attrs
---------------- -------- -------- -------- -------- -----
d                       0        2        3        5
dn                      0        0        3        3
g                       0        1        2        3
gnd                     0        1        0        1  sg
q                       0        4        3        7
qn                      0        2        3        5
-------------------------------------------------------
Total 6 nets            0       10       14       24
Maximum                 0        4        3        7
Average             0.000    1.667    2.333    4.000
```

The following example displays a verbose connection report for net x11.

```
nt_shell> report_net -connections -verbose x11
 ...
Connections for net 'xl1.g':
    Pin
    Capacitance: Min Rise Min Fall Max Rise Max Fall
    ------------------------------------------------
    data trace:      0.01     0.01     0.01     0.01
    clock trace:     0.01     0.01     0.01     0.01

    Wire
```

```
Capacitance: Min Rise Min Fall Max Rise Max Fall
-------------------------------------------------
data trace:      0.00     0.00     0.00     0.00
clock trace:     0.00     0.00     0.00     0.00

Total
Capacitance: Min Rise Min Fall Max Rise Max Fall
-------------------------------------------------
data trace:      0.01     0.01     0.01     0.01
clock trace:     0.01     0.01     0.01     0.01

Fanout Pins               Type
-------------------------------------------------
xl1.mn1.g                 Input Pin (transistor)

Inout Pins                Type
-------------------------------------------------
xck2.mp1.s                Inout Pin (transistor)
xck2.mn1.d                Inout Pin (transistor)
```

SEE ALSO

current_design(2)
current_instance(2)
report_cell(2)
report_design(2)

report_noise

Reports pins that have crosstalk noise data, sorted by the worst noise slack value on the pin.

SYNTAX

```
status report_noise
   [-above]
   [-below]
   [-high]
   [-low]
   [-nworst N]
   [-all_violators]
   [-data_pins]
   [-clock_pins]
   [-slack_lesser_than slack]
   [-height_greater_than height]
   [-significant_digits digits]
   [-nosplit]
   [-shape]
   [objects]
```

Data Types

```
N          integer
slack      float
height     float
digits     integer
objects    list
```

ARGUMENTS

-above

Reports noise that goes above the stable signal level (either high or low) of the victim net.

-below

Reports noise that goes below the stable signal level (either high or low) of the victim net.

-high

Reports noise values that occur when the victim net signal level is high.

-low

Reports noise values that occur when the victim net signal level is low.

-nworst *N*

The number of pins in the report. By default, a single pin is reported for each noise type.

-all_violators *N*

Reports all pins with negative noise slack values.

-data_pins

Reports all pins which are data inputs of clocked devices including latches and precharge cells.

-clock_pins

Reports all pins which are clock inputs of clocked devices including latches and precharge cells.

-slack_lesser_than *slack*

Reports all pins that have slack values less than the specified value.

-height_greater_than *height*

Reports all pins that have noise peak values greater than the specified value.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

-shape

Reports the width and time-to-peak ratio of noise bumps.

objects

A list of pins, ports, or nets to be reported. If no list is specified, all pins with noise values are considered.

DESCRIPTION

The **report_noise** command reports the pins that have crosstalk noise, listed according to the worst noise slack values found in the design. It can be used only after the **update_noise** command has been run, and only when crosstalk (signal integrity) noise analysis is enabled. You must have a NanoTime Ultra license to run crosstalk noise analysis.

You can select specific types of noise by using combinations of one or two of the **-above**, **-below**, **-high**, and **-low** options.

Use the **-all_violators** option to report all pins with noise slack values that are negative. Use the **-slack_lesser_than** option to select a set of pins whose noise slack value is less than the specified margin.

Use the **-data_pins** option to report noise at the input pins of register type cells such as latches, precharge cells, memories, or flip-flops. Use the **-clock_pins** option to report noise at the clock input of these same cells.

COMMAND PHASING

The **report_noise** command can be executed at any time after the **trace_paths** and **update_noise** commands have been run.

EXAMPLES

The following command reports the pins that have the 3 worst noise slack values:

```
nt_shell> report_noise -nworst 3
 ...

    slack    height type Pin              Net
--------- --------- ---- ---------------- ----------------
   -0.177     0.177 AH   x00.xbuf5.Mm_0.g x00.I2
   -0.177     0.177 AH   x00.xbuf5.Mm_1.g x00.I2
   -0.064     0.064 AH   x00.xbuf6.Mm_0.g x00.I3

   -0.192    -0.192 BH   x00.xbuf5.Mm_0.g x00.I2
   -0.192    -0.192 BH   x00.xbuf5.Mm_1.g x00.I2
   -0.070    -0.070 BH   x00.xbuf6.Mm_0.g x00.I3

   -0.154     0.154 AL   x00.xbuf5.Mm_0.g x00.I2
   -0.154     0.154 AL   x00.xbuf5.Mm_1.g x00.I2
   -0.042     0.042 AL   x00.xbuf6.Mm_0.g x00.I3

   -0.145    -0.145 BL   x00.xbuf5.Mm_0.g x00.I2
   -0.145    -0.145 BL   x00.xbuf5.Mm_1.g x00.I2
   -0.040    -0.040 BL   x00.xbuf6.Mm_0.g x00.I3
```

The following command reports pins that have noise slack values above the Vdd rail that are less than -.05 volts:

```
nt_shell> report_noise -above -high -slack_lesser_than -.05
 ...
    slack    height type Pin              Net
--------- --------- ---- ---------------- ----------------
   -0.177     0.177 AH   x00.xbuf5.Mm_0.g x00.I2
   -0.177     0.177 AH   x00.xbuf5.Mm_1.g x00.I2
   -0.064     0.064 AH   x00.xbuf6.Mm_0.g x00.I3
   -0.064     0.064 AH   x00.xbuf6.Mm_1.g x00.I3
```

SEE ALSO

update_noise(2)
report_noise_violation_sources(2)
report_fanout_noise(2)
si_enable_noise_analysis(3)

report_noise_coverage

Reports the coverage of injection noise analysis

SYNTAX

```
status report_noise_coverage
   [-coupling_capacitance]
   [-sum_of_effective_aggressors]
   [-number_of_effective_aggressors]
   [-exclude_nets]
   [-attribute_count]
   [-significant_digits N]
   [-nosplit]
```

Data Types

```
   N         integer
```

ARGUMENTS

-coupling_capacitance

Reports total coupling capacitance.

-sum_of_effective_aggressors

Reports total capacitance of effective aggressors.

-number_of_effective_aggressors

Reports total number of effective aggressors.

-exclude_nets

Excludes all the victims specified with the option.

-attribute_count

Reports the number of occurrences for each attribute. For example, if a victim has 3 aggressors and all of them are excluded for above low analysis, the attributes in the report include 'ALE(3)'.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

DESCRIPTION

The **report_noise_coverage** command generates a report about the coverage of injection noise analysis. The report includes all the top nets of each hierarchical group as victims along with other information, such as total coupling capacitance, sum of effective aggressor capacitance, number of effective aggressors and attributes for the noise analysis associated to the victims.

NanoTime does not report noise analysis results for some nets such as the ones with no timing point and internal nets of a cluster. The noise coverage helps to identify such cases. In the example below, noise heights for net bbgclk123 are missing because the victim net doesn't have any gate pins on it (NGP in the attributes column).

The report is also helpful to track the conditions of the aggressors and the victim for noise analysis. From the example below, you can identify that the victim net n1clk11 has static logic state high (LS1) and aggressor exclusion for above low (ALE) and for below high (BHE) analysis. The attributes reflect not only user specified exclusions and markings but also internal exclusions and markings.

Note that when one attribute covers the other attributes, only the covering attribute is reported. For example, if a victim doesn't have coupling capacitance (NC), no other attributes are printed but NC. Similarly, if an aggressor is excluded for above low analysis (ALE), LA1 or LA0 is not printed even though the aggressor has static logic. The list of all the attribute codes and explanations is printed in the legend of the report.

COMMAND PHASING

The **report_noise** command can be executed at any time after the **update_noise** command has been run successfully.

EXAMPLES

The following command reports the coverage of noise analysis.

```
nt_shell> report_noise_coverage -coupling_cap -sum_of_effective_aggr \
-number_of_effective_aggr -significant_digits 6
 ...

          Sum Of
```

```
 Sum Of     Eff.
Coupling    Aggr.   # Of Eff.
 Cap.       Cap.     Aggr.   BH Height AL Height Attributes                Victim
-------- --------- --------- --------- --------- ------------------------ ---------
0.010200  0.010200       6     -0.193381 0.191487   BHI ALI G2C TRI          bD0
0.010200  0.010200       6     -0.193381 0.191487   BHI ALI G2C TRI          bD1
0.006300  0.006300       5     -0.002235 0.004902   BHI ALI PFB              bout1
0.004000  0.004000       2         -         -      BHI ALI NGP              bbgclk1
0.008000  0.008000       4         -         -      ALE BHE PUC PUG C2G LS1  n1clk13
```

SEE ALSO

si_enable_noise_analysis(3)
update_noise(2)
report_noise(2)

report_noise_violation_sources

Reports the aggressor nets that contribute to the noise reported on a victim pin.

SYNTAX

```
status report_noise_violation_sources
   [-above]
   [-below]
   [-high]
   [-low]
   [-nworst N]
   [-max_aggressors N]
   [-aggressor_windows]
   [-aggressor_contributions]
   [-net]
   [-significant_digits digits]
   [-nosplit]
   [objects]
```

Data Types

```
N           integer
digits      integer
objects     list
```

ARGUMENTS

-above

Reports aggressors that contribute to noise that goes above the stable signal level of the victim net (either high or low).

-below

Reports aggressors that contribute to noise that goes below the stable signal level of the victim net (either high or low).

-high

Reports aggressors that contribute to noise on the high signal level of the victim net (such as vdd).

-low

Reports aggressors that contribute to noise on the low signal level of the victim net (such as ground).

-nworst *N*

The number of victim pins to be reported. By default, one victim pin is reported.

-max_aggressors *N*

The number of aggressors to be reported for each victim pin. By default, one aggressor is reported.

-aggressor_windows

Displays the aggressor windows.

-aggressor_contributions

Displays the aggressor contributions.

-net

Displays the noise victim net name. By default only the victim pin name is displayed.

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

objects

A list of pins, ports, or nets to be reported. If no list is specified, all pins with noise values are considered.

DESCRIPTION

The **report_noise_violation_sources** command reports the sources of noise for pins that have noise values. For each aggressor, the report includes the coupling capacitance to the victim, the percentage of the total coupling capacitance to the victim, and the minimum rising and falling transitions.

The victims are sorted by worst noise slack. The aggressors for each victim are sorted by the largest coupling capacitance value. To get the aggressor windows, specify the **-aggressor_windows** option. To report estimates of the percentage contributions to the total noise from each aggressor, specify the **-aggressor_contributions** option.

Noise bumps that have been defined using the **set_input_noise** command are also shown as sources.

If coupling capacitance variation has been enabled, the column for an aggressor's coupling capacitance displays a range, as in **[min_cap, max_cap]**. The column for an aggressor's percentage of the total

coupling capacitance continues to be based on nominal coupling capacitance values.

Specific types of noise can be selected by using the **-above**, **-below**, **-high**, and **-low** options.

The timing windows reported by the **-aggressor_windows** option appear in one of the following formats. In this table, *min* represents the smallest minimum delay arrival time, *max* represents the largest maximum delay arrival time, *clock* is the name of the associated clock, and *dir* is either R or F for the transition direction of the clock reference edge.

```
INF                  No timing window available; NanoTime uses an
                     infinite timing window.

(min max)            Timing window for a fully-specified non-periodic
                     non-clock domain.

(INF INF)            Aggressor is part of a non-periodic non-clock
                     domain, but no timing window is available.
                     NanoTime uses an infinite timing window.

(INF max)            Aggressor is in a non-periodic non-clock domain,
                     but the minimum delay arrival time is unknown.

(min INF)            Aggressor is in a non-periodic non-clock domain,
                     but the maximum delay arrival time is unknown.

(min max clock dir)  Timing window for a fully-specified periodic clock
                     domain.

(INF INF clock dir)  Aggressor is in the specified clock domain, but no
                     timing window is available; NanoTime uses an
                     infinite timing window.

(INF max clock dir)  Aggressor is in the specified clock domain,
                     but the minimum delay arrival time is unknown.

(min INF clock dir)  Aggressor is in the specified clock domain,
                     but the maximum delay arrival time is unknown.
```

COMMAND PHASING

The **report_noise_violation_sources** command can be executed any time after the **update_noise** command.

EXAMPLES

The following command reports the sources of noise for the three worst noise victims. The report displays the two aggressors with largest coupling capacitance values for each victim.

```
nt_shell> report_noise_violation_sources -nworst 3 -max_aggressors 2
 ...

 coup  pct of   aggressor  noise                  aggressor aggressor
  cap   total       trans  slack type Victim pin   net       window
----- -------  ---------- ------ ---- ------------ -------- -------------
0.016   50.07  0.438 (F)  -0.131  BL  xbuf5.Mm_0.g  x00.I1  (1.175 1.180)
```

```
0.016   49.88  0.442 (F)  -0.131  BL  xbuf5.Mm_0.g  x00.I3  (1.174 1.182)
0.016   50.07  0.438 (F)  -0.131  BL  xbuf5.Mm_1.g  x00.I1  (1.175 1.180)
0.016   49.88  0.442 (F)  -0.131  BL  xbuf5.Mm_1.g  x00.I3  (1.174 1.182)
0.016   99.78  1.147 (F)  -0.032  BL  xbuf6.Mm_0.g  x00.I2  (1.424 1.446)
0.000    0.22  0.050 (FD) -0.032  BL  xbuf6.Mm_0.g  Z3      (INF INF)
```

SEE ALSO

```
update_noise(2)
report_noise(2)
set_input_noise(2)
si_enable_noise_analysis(3)
```

report_paths

Reports the timing of paths found during path tracing.

SYNTAX

```
status report_paths
   -min | -max | -path_id path_id_list | objects
   [-clock_only]
   [-by slack | -by delay]
   [-show_path_id]
   [-show_path_index]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-slack_less_than slack_limit]
   [-slack_greater_than slack_limit]
   [-nworst paths_per_endpoint]
   [-npaths_per_startpoint paths_per_startpoint]
   [-max_paths max_path_count]
   [-suppress_similar_paths]
   [-path_type format]
   [-nets]
   [-transition_time]
   [-full_transition_time]
   [-variation]
   [-coefficient_adjustment]
   [-coefficient_ratio]
   [-capacitance]
   [-wire_delay]
   [-crosstalk_delta]
   [-rail_voltage]
   [-final_voltage]
   [-significant_digits digits]
   [-nosplit]
```

Data Types

```
   path_id_list            list
   objects                 list
   from_list               list
   rise_from_list          list
   fall_from_list          list
   through_list            list
```

```
rise_through_list        list
fall_through_list        list
to_list                  list
rise_to_list             list
fall_to_list             list
slack_limit              float
paths_per_endpoint       integer
paths_per_startpoint     integer
max_path_count           integer
format                   string
digits                   integer
```

ARGUMENTS

-min

Reports minimum delay (shortest) paths only. One of the **-min**, **-max**, **-path_id** options or the *objects* argument must be used.

-max

Reports maximum delay (longest) paths only.

-path_id *path_id_list*

Specifies the ID numbers of the paths to report. This option cannot be used at the same time as the **-min**, **-max**, or *objects* options.

objects

Specifies a collection of timing path objects to report. This option cannot be used at the same time as the **-min**, **-max**, or **-path_id** options. You can create a collection of timing paths with the **get_timing_paths** command.

-clock_only

Reports clock paths only. A clock path starts at a clock input pin and ends at the clock pin of a sequential device. Use the **-by delay** option if you want the clock paths reported in order of increasing delay (with the **-min** option) or decreasing delay (with the **-max** option).

-by slack

Lists the paths in order by slack (the default). Paths with the least slack are listed first.

-by delay

Lists the paths in order by delay. Paths with the worst delay are listed first.

-show_path_id

Shows the path ID number of each path in the report. For example, you can generate a summary report that lists many paths with their ID numbers by using the **-show_path_id** option, and later request a detailed report on a single path by specifying its ID number with the **-path_id** option.

-show_path_index

Shows an index number at the beginning of each path report. The path reports are numbered in sequence: 1, 2, 3, 4, and so on. This feature is useful for a long report containing many paths. The index number of the last report indicates the total number of paths in the report.

-from ***from_list***

Specifies a list of pins, ports, or nets to be considered path startpoints. Only paths that start from the specified objects are reported.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only rising-edge transitions from the specified objects are considered.

-fall_from ***fall_from_list***

Same as the **-from** option, except that only falling-edge transitions from the specified objects are considered.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass in order to be reported. If you specify multiple objects in a *through_list* for a single **-through** option, a path can pass through any one of those objects. If you use multiple **-through** options, a path must pass through each **-through** object in sequential order.

-rise_through ***rise_through_list***

Same as the **-through** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_through ***fall_through_list***

Same as the **-through** option, except that only falling-edge transitions arriving at the specified objects are considered.

-to ***to_list***

Specifies a list of pins, ports, or nets to be considered path endpoints. Only paths that end at the specified objects are reported.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_to ***fall_to_list***

Same as the **-to** option, except that only falling-edge transitions arriving at the specified objects are considered.

-slack_less_than ***slack_limit***

Restricts reporting to paths that have a slack less than (more negative than) the specified value. Specify 0.0 to report only the paths that have timing violations.

-slack_greater_than ***slack_limit***

Restricts reporting to paths that have a slack greater than (less negative than) the specified value.

-nworst ***paths_per_endpoint***

Specifies the maximum number of paths reported per endpoint. By default, there is no limit.

-npaths_per_startpoint ***paths_per_startpoint***

Specifies the maximum number of paths reported per startpoint. By default, there is no limit.

-max_paths ***max_path_count***

Specifies the maximum total number of paths to be reported. By default, there is no limit and all paths in the design are reported.

-suppress_similar_paths

Suppresses the display of multiple paths that are similar. When two or more paths are similar, only the first path is reported, along with a notation showing the number of similar paths that are not being reported. 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.

-path_type ***format***

Specifies the path reporting format. Valid values are **full_clock_expanded**, **full_clock**, **full**, **summary**, **short**, **list**, and **end**. The default is **list**, which lists the paths, one path per line, showing the startpoint, endpoint, delay, and slack of each path.

-nets

Adds net name information to a full_clock_expanded, full_clock, full, or summary report.

-transition_time

Adds transition time information to a full_clock_expanded, full_clock, full, or summary report.

-full_transition_time

Adds transition time information to a full_clock_expanded, full_clock, full, or summary report, with the transition time linearly extrapolated to ground and Vdd. This option is useful when the **-rail_voltage option** option is also used.

-variation

Report the individual stage variations during a full_clock_expanded, full_clock, or full report. The variation is from the use of the -pocv option to trace_paths. The detailed report will show the variation computed at each stage of the path.

-coefficient_adjustment

Report the adjustment to the stage delay that is caused by user defined coefficients specified with the set_delay_coefficient command. The change in delay from a non-adjusted delay will be reported in a separate column, indicating the impact of user defined delay adjustment settings on each stage delay.

-coefficient_ratio

Report the adjustment to the stage delay that is caused by user defined coefficients specified with the set_delay_coefficient command. The change in delay from a non-adjusted delay will be reported as the ratio of (non-adjusted delay + delay adjustment) / non-adjusted delay.

-capacitance

Adds net capacitance information to a full_clock_expanded, full_clock, full, or summary report.

-wire_delay

Adds wire delay information to a full_clock_expanded, full_clock, full, or summary report. You must set the **timing_save_wire_delay** variable to **true** before path tracing to save the wire delay information for reporting. This might cause a significant increase in memory use and possibly runtime.

-crosstalk_delta

Adds crosstalk delta delay information to a full_clock_expanded, full_clock, full, or summary report when crosstalk analysis is enabled.

-rail_voltage

Adds rail voltage information to a full_clock_expanded, full_clock, full, or summary report.

-final_voltage

Adds final voltage information to a full_clock_expanded, full_clock, full, or summary report.

-significant_digits ***digits***

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable, whose default is 3.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

DESCRIPTION

The **report_paths** command generates a report on timing paths in the design. The command options let you specify report parameters such as check type (minimum or maximum delay), path ordering by worst-case slack or delay, the number of paths reported, the path scope (startpoints, throughpoints, and endpoints), and the level of detail to include.

The **report_paths** command only produces meaningful output after path tracing is complete. The **report_paths** command gets the timing information from the database without performing any additional path tracing or simulation.

In the **report_paths** command, you must specify which paths to report. One approach is to use the **-min** or **-max** options to specify minimum-delay (shortest) or maximum-delay (longest) paths.

Alternatively, you can specify the paths to report by ID number. To do this, first generate a summary report using the **report_paths** command with the **-min** or **-max** option and the **-show_path_id** option, and without the **-path_type** option. The result is a list of paths, one line per path, showing the ID number, delay, and slack. When you find the path of interest in the list, you can then get a detailed report on that path by using the **report_paths** command with the **-path_id** option to specify the ID number.

Path ID numbers are generated during any path-tracing operation (in other words, after the **trace_paths**, **extract_model**, or **characterize_context** commands). ID numbers are assigned only to the worst-case paths saved in the path database. The numbers are arbitrary and are not ordered in any manner. The ID number assigned to a particular path might change with each path-tracing operation.

To get a report on clock paths only and skip reporting of the data paths, use the **-clock_only** option. Use the **-by delay** option if you want the clock paths reported in order of increasing delay (with the **-min** option) or decreasing delay (with the **-max** option). Otherwise, the clock paths are reported in order of increasing slack, taking into consideration the associated data paths.

By default, the **report_paths** command reports paths in order of increasing slack, starting with the path having the least (or most negative) slack. To list paths in order of delay instead, starting with the path having the worst delay, use the **-by delay** option. The **-by delay** option reports unconstrained paths (for example, a path to an output that has no output delay set), as well as constrained paths.

Number of Paths Reported

By default, the **report_paths** command reports all timing paths in the design, typically resulting in a very long report. To reduce the runtime for a long report, and to keep a record of the output, you can redirect the command output to a file. For example:

```
nt_shell> report_paths -max > myfile.txt
```

If you are viewing the command output directly on the screen, and the report is extremely long, you can interrupt the command output by typing Ctrl+C. To scroll through longer reports one screenful at a time (like using the **more** command under UNIX), set the **sh_enable_page_mode** variable to **true**.

For viewing reports on the screen, you should reduce the output by using the **-max_paths** option. For example, the following command reports only the 10 worst-case maximum-delay paths:

```
nt_shell> report_paths -max -max_paths 10
```

To limit the report to paths having a slack worse than a specified value, use the **-slack_less_than** option. For example, the following command reports only the paths that have a negative slack (paths with timing violations):

```
nt_shell> report_paths -max -slack_less_than 0.0
```

Many paths might share a common endpoint. By default, the **report_paths** command reports all of these paths. To reduce the number of paths reported per endpoint, use the **-nworst** option. For example, the following command reports only the single worst maximum-delay path per endpoint:

```
nt_shell> report_paths -max -nworst 1
```

To suppress the reporting of similar paths, use the **-suppress_similar_paths** option. In that case, when two or more paths are similar, only the first path is reported, along with a notation showing the number of similar paths that are not reported. This results in a much shorter report when there are many similar paths (for example, designs with buses).

A path is considered similar to another path if both of the following conditions are true:

1. The path has at least one net in the same group as the corresponding net in the other path. Net groups are defined with the **create_net_group** command.
2. The total path delays, slacks, and stage delays of the two paths are similar.

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 time units. The "relative" settings specify the allowable percentage difference.

Specifying the Scope (-from, -to, -through)

You can optionally restrict the scope of paths reported based on the path startpoints, throughpoints, and endpoints. To do so, use the **-from**, **-through**, and **-to** type options.

However, the **-through** option might return unexpected results because the specified paths might not have been saved into the paths database during the path tracing operation. Even if some paths are reported by using the **report_paths -through** command, there is no guarantee that those paths represent the worst delays through that point.

To force NanoTime to perform path tracing through a specified point, use one or more **set_find_path** commands with the **-through** option before you execute the **check_design** command. You can then use the **report_paths** command to examine the delays on those paths. In this case, path tracing is performed only on the specified paths. Delays in the rest of the design are not saved in to the path database and are therefore not available for reporting.

For more information on using the **-from**, **-through**, and **-to** type options, see the man page for the **set_find_path** command or the *NanoTime User Guide.*

Path Type Report Formats

You can choose the level of detail shown in the path timing reports by using the **-path_type** option. These are the available option settings, listed in order from most detailed to least detailed reporting:

```
Report type            Information shown in report

full_clock_expanded   data paths include chained generated clock paths
                            clock paths include chained generated clock paths and slack
full_clock            data path, clock path, slack
full                  data path, slack
summary               data path
short                 start/end/slack calculation
list                  start/end/slack in one line
end                   end/slack in one line
```

The default path reporting format is **list**, which generates a list of paths, one line per path. Each line shows the path startpoint, endpoint, timing path type, path delay, and slack. For example:

```
                                    Path   Path
   Startpoint    Endpoint           Type   Delay    Slack
------------ ---------------------- ------ ------- ------
r  CK         f  Xs.Xreg50.X3.Mp0.g  C-L    4.217   -1.160
r  clk2       f  Xs.Xreg51.X3.Mp0.g  C-L    3.467   -1.160
r  clk2       f  Xs.Xreg52.X3.Mp0.g  C-L    3.467   -1.160
r  CK         f  Xs.Xreg51.X3.Mp0.g  C-L    4.217   -1.160
r  clk2       f  Xs.Xreg50.X3.Mp0.g  C-L    3.467   -1.159
```

The notation "r" means rising edge. The "Path Type" column shows the type of timing point at the start and end of the path. In this example, "C-L" indicates a clock-to-latch path. The following abbreviations are used to describe different net types:

```
D     - Data input
O     - Data output
SZ    - Data turnoff output
Z     - Turnoff
E     - Turnoff enable
C     - Clock
CX    - Clock with dynamic simulation
L     - Latch
R     - Register File Latch
A     - Adjusted latch
RA    - Adjusted Register File latch
G     - Gated clock
T     - Transparent gated clock
SC    - Stopped clock
U     - User-defined constraint
DU    - User-defined data-to-data constraint
M     - Timing model
LP    - Latch to latch loop
PC    - Precharge Clock
EC    - Eval Clock
PP    - Precharge PC and Predischarge EC
PN    - Precharge EC and Predischarge PC

N1    - Precharge node of D1 N-domino
N2    - Precharge node of D2 N-domino
N3    - Precharge node of D1 N-domino retain
N4    - Precharge node of D2 N-domino retain
N5    - Precharge node of D1 N-domino latch
N6    - Precharge node of D2 N-domino latch
N7    - Precharge node of D1 N-domino flip-flop
N8    - Precharge node of D2 N-domino flip-flop

n1    - Input of D1 N-domino
n2    - Input of D2 N-domino
n3    - Input of D1 N-domino retain
n4    - Input of D2 N-domino retain
n5    - Input of D1 N-domino latch
n6    - Input of D2 N-domino latch
n7    - Input of D1 N-domino flip-flop
n8    - Input of D2 N-domino flip-flop

P1    - Predischarge node of D1 P-domino
P2    - Predischarge node of D2 P-domino
P3    - Predischarge node of D1 P-domino retain
P4    - Predischarge node of D2 P-domino retain
P5    - Predischarge node of D1 P-domino latch
P6    - Predischarge node of D2 P-domino latch
P7    - Predischarge node of D1 P-domino flip-flop
P8    - Predischarge node of D2 P-domino flip-flop

p1    - Input of D1 P-domino
p2    - Input of D2 P-domino
p3    - Input of D1 P-domino retain
p4    - Input of D2 P-domino retain
p5    - Input of D1 P-domino latch
p6    - Input of D2 P-domino latch
p7    - Input of D1 P-domino flip-flop
p8    - Input of D2 P-domino flip-flop

GC    - Generated clock
```

```
M1-16 - Memory-specific timing checks
$     - Simultaneous switching inputs (see report_paths man page)
&     - Net has backannotated parasitics
```

The list refers to type D1 and type D2 domino precharge structures. A type D1 domino structure has a controlling clock on the evaluate stack, whereas a type D2 domino structure does not.

The list also refers to N-domino and P-domino circuits. An N-domino circuit has an NMOS evaluate stack. This type of circuit charges a precharge node to a high voltage during a precharge phase, when the clock is low; and then conditionally discharges that node during the evaluate phase, when the clock is high.

A P-domino circuit has a PMOS evaluate stack. This type of circuit discharges a predischarge node to a low voltage during a predischarge phase, when the clock is high; and then conditionally charges that node during the evaluate phase, when the clock is low.

A **full_clock_expanded** report shows the full data and clock paths starting from an input port including all generated clock segments on the paths, if any. The report also shows the incremental and cumulative delay values at intermediate points within the data path and clock path, followed by the slack calculation. For a maximum-delay path, the slack is the data required time minus the data arrival time. For a minimum-delay path, the slack is the data arrival time minus the data required time. This option does not report paths expanding forward. The **-from** point must be the last generated clock in the path and the **-through** point must be downstream from the last generated clock point.

A **full_clock** report shows the incremental and cumulative delay values at intermediate points within the data path and clock path, followed by the slack calculation. For a maximum-delay path, the slack is the data required time minus the data arrival time. For a minimum-delay path, the slack is the data arrival time minus the data required time.

A **full** report is like a **full_clock** report, except that the intermediate points within the clock path are not shown. A **summary** report is like a **full** report, except that it shows only the data path timing. The clock timing and slack are not shown.

For a **full_clock_expanded**, **full_clock**, **full**, or **summary** report, you can display the net names, transition times, full-swing transition times, net capacitance, and wire delay values in the detailed path reports by using the **-nets**, **-transition_time**, **-full_transition_time**, **-capacitance**, and **-wire_delay** options. By default, these types of information are not shown in the reports.

All full reports list the PBSA common net (or pin) in the header, if PBSA is applicable between the launch and capture paths.

To get other types of information from the path database in the form of object attributes, use the **get_timing_paths** command.

COMMAND PHASING

The **report_paths** command can only be executed after path tracing is complete.

EXAMPLES

The following command reports all maximum-delay paths in the design. The report lists the paths in order,

starting with paths having the least slack, one line per path. The report can be very long. To interrupt a long report in progress, type Ctrl+C.

```
nt_shell> report_paths -max
 ...
             Path  Path
   Slack    Delay  Type    Startpoint        Endpoint
-------- -------- ----- -- -------------- -- --------------------
   0.346    4.154   C-O r  clk2           r  SBUS[63]
   0.346    4.154   C-O r  clk2           r  SBUS[62]
   0.385    4.115   C-O r  clk2           r  SBUS[61]
   0.410    4.090   C-O r  clk2           r  SBUS[60]
   0.434    4.066   C-O r  clk2           r  SBUS[59]
   0.517    3.983   C-O r  clk2           f  SBUS[63]
   0.517    3.983   C-O r  clk2           f  SBUS[62]
   0.541    3.959   C-O r  clk2           r  SBUS[58]
   0.547    4.131   C-L r  clk2           f  Xareg.Xreg62.X5.Mp0.G
   0.547    4.131   C-L r  clk2           f  Xbreg.Xreg62.X5.Mp0.G
 ...
```

The following command redirects the output to a file called myfile.txt. For a long report, this runs much faster than viewing the output on the screen.

```
nt_shell> report_paths -max > myfile.txt
```

The following command reports only the maximum-delay paths that end at port BUS[0].

```
nt_shell> report_paths -max -to [get_ports BUS[0]]
```

The following command reports the 20 worst-case (least-slack) maximum-delay paths in the design.

```
nt_shell> report_paths -max -max_paths 20
```

The following command reports all the minimum-delay paths in the design that have a negative slack.

```
nt_shell> report_paths -min -slack_less_than 0.0
```

The following example generates a detailed report on the single worst-slack, maximum-delay path in the design. The report shows the intermediate points along the path, the type of net, the incremental delay for that point, any delay adjustment (for example, latch error recovery), and the cumulative path delay up to that point. At the end of the report is the slack calculation, the data required time minus the data arrival time.

Each entry in the Point column shows either an incremental point along the timing path or a value used in the slack calculation such as the external delay, setup time, clock uncertainty, data required time, or data arrival time. An entry that says "delta to checked pin" indicates the time difference between the arrival time at the checked pin of a timing constraint and the endpoint of the path that caused the constraint to be evaluated. This difference can occur for a setup check to a latch output or a hold check to a latch input, for example.

```
nt_shell> report_paths -max -max_paths 1 \
          -path_type full -nets


Net Types (NT):
    D    - Data input
    O    - Data output
    SZ   - Data turnoff output
 ...
Startpoint:        clk2 (in port)
```

```
Endpoint:          SBUS[63] (out port)
Path Type:         max
Constraint:        set_output_delay check

  Path     Incr   Adjust NT    Point                          Net
------ -------- -------- -- -- ------------------------------ --------
 1.500             1.500 C    r clk2 (in)                    clk2
 1.500    0.000          C    r Xsub.Xadder.X0.Mn0.G (inv2)  clk2
...
 5.213    0.243               r Xsub.Xadder.X6.X0.Mn0.G (inv)  Xsub...
 5.265    0.051               f Xsub.Xadder.X6.X1.Mp0.G (inv)  Xsub...
 5.400    0.136               r Xlc63.X5.Mn0.G (inv3)          Xlc63.S
 5.484    0.083               f Xls63.X5.Mp0.G (inv2)          Xls63.Sn
 5.549    0.065          E    r Xsreg.Xreg63.Mn1.G (latch)     S[63]
 5.626    0.077          Z    f Xsreg.Xreg63.X6.Mp0.G (inv4)   Xsreg...
 5.654    0.028          O    r SBUS[63] (out)                 SBUS[63]
 5.654                         data arrival time
          4.154    1.500       Total

 6.000    6.000               clock MCLK (rise)
 6.000    0.000               output external delay
 6.000    0.000               clock uncertainty
 6.000                        data required time
-----------------------------------------------------------------------
 6.000                        data required time
-5.654                        data arrival time
-----------------------------------------------------------------------
 0.346                        slack (MET)
```

SEE ALSO

create_net_group(2)
create_timing_check(2)
get_attribute(2)
get_timing_paths(2)
reset_design(2)
set_find_path(2)
trace_paths(2)
report_paths_suppress_similar_paths_delay_absolute_tolerance(3)

report_pbsa_calculation

Generates a report on the path-based slack adjustment (PBSA) calculations for a path obtained with the **get_timing_paths** command.

SYNTAX

```
status report_pbsa_calculation
   [-significant_digits digits]
   [path_list]
```

Data Types

```
   digits          integer
   path_list       list
```

ARGUMENTS

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

path_list

A list of paths generated by the **get_timing_paths** command.

DESCRIPTION

The **report_pbsa_calculation** command generates a report of the calculations used in path-based slack adjustment (PBSA). Path-based slack adjustment is a NanoTime feature that adjusts the worst-case delay of each path segment by an amount proportional to the length of the path segment. Compared with using the **set_clock_uncertainty** command to set global uncertainty values, using PBSA results in less pessimism, but requires more effort and runtime. You must have a NanoTime Ultra license to use PBSA.

To calculate the delay adjustment for each path segment, NanoTime uses the number of simulation levels (channel-connected block units) and the originally calculated delay, in conjunction with user-specified scaling factors set through NanoTime application variables.

To use path-based slack adjustment in place of fixed worst-case uncertainty settings, do the following:

1. Reduce or eliminate the global uncertainty values previously set with the **set_clock_uncertainty** command.

2. Set the variables that control the path adjustment calculations.

3. Use the **-pbsa** option with the **trace_paths** command.

Invoking PBSA changes the timing results, so the paths with the worst-case timing can be different from the worst-case paths without PBSA. By default, the **trace_paths** command saves only the single worst-case path per startpoint-endpoint pair. Therefore, if you want to compare the timing of a path with and without PBSA, you might need to save more paths in the path database. To increase the number of paths saved by **trace_paths**, set the **-keep_paths_within** option to a value slightly larger than the default of 0.0 and set the **-npaths** option to a value larger than 1.

After path tracing is completed by the **trace_paths** command, the slack values reported by the **report_paths** command reflect the adjustments made by PBSA. To find out the size of the PBSA adjustment, use the option **-path_type full** or **-path_type full_clock** in the **report_paths** command.

The **report_pbsa_calculation** command generates a report that shows the variable settings, the original delay and number-of-levels parameters of the path segments, and the calculated components of the total PBSA slack adjustment. The **report_pbsa_calculation** command operates on one or more paths obtained with the **get_timing_paths** command, which creates a collection of paths taken from the paths stored in the path database.

You can set a variable to a collection created by the **get_timing_paths** command, and then use the **report_pbsa_calculation** command to get information about the paths in the collection. For example:

```
nt_shell> set path1 [get_timing_paths -max -max_paths 3]
nt_shell> report_pbsa_calculation $path1
```

You can change the PBSA variable settings and run the report again to see the changes to the adjustment values. However, this does not affect the path database or the path reports generated by the **report_paths** command. To update the path database and path reports with new PBSA variable settings, use the **reset_design -paths** command, followed by the **trace_paths -pbsa** command.

Path Segments

For path-based slack adjustment, NanoTime considers four different path segments in each setup or hold timing check. These path segments are designated A, B, C, and D. Different scaling factors apply to each path segment.

Segment A is the shared or common portion of the launch clock path and capture clock path (if any) from the clock input port to the common point.

Segment B is the portion of the launch clock path from the common point to the clock pin of the launching sequential element.

Segment C is the portion of the capture clock path from the common point to the clock pin of the capturing sequential element.

Segment D is the datapath, from the clock pin of the launching sequential element to the data input pin of the capturing sequential element.

If the launch and capture clock paths share a common segment from the clock source, that segment is called segment A, and the point at which the launch and capture clock paths diverge is called the common point. If there is no shared segment, the common point is at the clock input port and there is no segment A.

Variables that Control PBSA Analysis

NanoTime uses a set of application variables to adjust the delays for each path segment. For details about the PBSA calculations, see the NanoTime User Guide.

The first section of the PBSA calculation report lists the user-specified values of the PBSA variables. Path-related variable names indicate the path segment (A, B, C, or D), the path type (minimum delay or maximum delay), and the scaling factor type.

The NanoTime variable names begin with the prefix pbsa_, followed by the label shown in the report (see the report in the Examples section). For example, the value listed as KAmin in the report is defined by the **pbsa_KAmin** variable. The values reported as KAceilmin, KBceilmin, and so on come from the **pbsa_KAceilingmin** and similar variables. The values reported for the *floor* and *ceil* variables values might not be the same as the values you set in the associated variables because the analysis might reset the values; see the NanoTime User Guide for details.

Slack Adjustment Calculation for Setup and Hold Checks

For a setup check in general:

```
Setup Slack = (Required Time) - (Arrival Time) - (Clk Unc)
```

For a conventional setup check (without PBSA), slack is calculated as follows:

```
Setup Slack = [A(min)+C(min)+Per] - [A(max)+B(max)+D(max)] - (Clk Unc)
```

For a hold check in general:

```
Hold Slack = (Arrival Time) - (Required Time) - (Clk Unc)
```

For a conventional hold check (without PBSA enabled):

```
Hold Slack =  [A(min)+B(min)+D(min)] - [A(max)+C(max)] - (Clk Unc)
```

A(min) is the minimum delay of path segment A and A(max) is the maximum delay, with similar notation for the other segments. Per is the clock period, where capture occurs on the next clock edge after the launch event. Clk Unc is the clock uncertainty value set with the **set_clock_uncertainty** command.

The analysis would have clock reconvergence pessimism to the extent that A(min) and A(max) are different. However, the delay through the common segment (segment A) must be a single value during a single timing check; therefore the A(min) and A(max) terms in the equation cancel. For this reason, the variables pbsa_KA* do not affect PBSA analysis, but are available for completeness.

Simple PBSA Analysis

The simplest PBSA analysis uses a multiplier for the path delay (written here as B(min) and so on) and another multiplier for the number of levels (written here as BL(min) and so on). For a setup check with PBSA enabled:

```
Slack =  + (Adjusted Required Time)
         - (Adjusted Arrival Time)
         - (Adjusted Clock Uncertainty)

Adjusted Required Time =
         + Period
         + [ KCmin + KCmin*C(min) + KCLmin*CL(min) ]

Adjusted Arrival Time =
         + [ KBmax + KBmax*B(max) + KBLmax*BL(max) ]
         + [ KDmax + KDmax*D(max) + KDLmax*DL(max) ]

Adjusted Clock Uncertainty =
         + [ KUsetup*(Clock Uncertainty) ]

Combining equations:

Setup Slack = + Period
              + [ Cmin + KCmin*C(min) + KCLmin*CL(min) ]
              - [ Bmax + KBmax*B(max) + KBLmax*BL(max) ]
              - [ Dmax + KDmax*D(max) + KDLmax*DL(max) ]
              - [ KUsetup*(Clock Uncertainty) ]

A similar equation arises for hold checks:

Hold Slack  = + [ Bmin + KBmin*B(min) + KBLmin*BL(min) ]
              + [ Dmin + KDmin*D(min) + KDLmin*DL(min) ]
              - [ Cmax + KCmax*C(max) + KCLmax*CL(max) ]
              - [ KUhold*(Clock Uncertainty) ]
```

PBSA Analysis with Level-Based Derating

The KBLpct* type factors provide a more flexible method to take the number of simulation levels into account. These factors modify the KB* type factors based on the number of levels. For details, see the NanoTime User Guide.

For example, with level-based derating you can choose to increase the maximum delay of path segment B by 10 percent for a one-stage path, 9.75 percent for a two-stage path, 9.5 percent for a three-stage path, and so on down to a minimum adjustment of 5.0 percent.

COMMAND PHASING

The **report_pbsa_calculation** command must be executed after path tracing is complete.

EXAMPLES

The following command generates a PBSA report for the worst-case (least-slack) hold check stored in the path database.

```
nt_shell> report_pbsa_calculation [get_timing_paths -max -max_paths 1]
```

```
****************************************
Report : pbsa calculation
Design : ALU
Version: H-2012.12
Date   : Tue Feb  5 13:53:05 2013
****************************************
Param      Value   Param      Value   Param      Value   Param     Value
---------- ------- ---------- ------- ---------- ------- --------- ------
KAmin      -0.500  KBmin      -0.500  KCmin      -0.500  KDmin     -0.500
KAmax      1.200   KBmax      1.200   KCmax      1.200   KDmax     1.200
KALmin     0.000   KBLmin     0.000   KCLmin     0.000   KDLmin    0.000
KALmax     0.000   KBLmax     0.000   KCLmax     0.000   KDLmax    0.000
KALpctmax  -0.100  KBLpctmax  -0.100  KCLpctmax  -0.100  KDLpctmax -0.100
KALpctmin  0.200   KBLpctmin  0.200   KCLpctmin  0.200   KCLpctmin 0.200
KAfloormax 0.500   KBfloormax 0.500   KCfloormax 0.500   KDfloormax 0.500
KAceilmin  -0.100  KBceilmin  -0.100  KCceilmin  -0.100  KDceilmin -0.100
KUsetup    1.000   KUhold     1.000

COMPUTED PATH PARAMETERS
Param       Value     Param        Value
----------- --------- ------------ -----
A clk delay 0.000     A clk levels   0
Adelay      0.000     Alevels        0
Bdelay      1.513     Blevels        5
Cdelay      0.888     Clevels        4
Ddelay      22.849    Dlevels       78

ADJUSTMENT CALCULATIONS
Adjustment   Value
------------ ---------
type         setup
common mode  0.000
uncertainty  0.000
clock path   0.711
data path    45.684
total slack  -44.973
```

The following command generates a PBSA report for the worst-case (least-slack) setup check stored in the path database that starts at port IN1 and ends at port OUT2.

```
nt_shell> report_pbsa_calculation [get_timing_paths \
          -from [get_ports IN1] \
          -to [get_ports OUT2] \
          -max -max_paths 1]
```

SEE ALSO

```
get_timing_paths(2)
trace_paths(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_KALpctmin(3)
pbsa_KAfloormax(3)
pbsa_KAceilingmin(3)

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

report_pin_variation

Reports the pin variations specified for the delay arcs associated with the specified trigger objects.

SYNTAX

```
status report_pin_variation
   [-min]
   [-max]
   [-rise]
   [-fall]
   [-voltage supply_voltage]
   [-target target_objects]
   trigger_objects
```

Data Types

```
   supply_voltage          float
   target_objects          list
   trigger_objects         list
```

ARGUMENTS

-min

Reports variation values for short path analysis.

-max

Reports variation values for long path analysis.

-rise

Reports variation values for rising triggers.

-fall

Reports variation values for falling triggers.

-voltage *supply_voltage*

Restrict the report to only variation values that apply to the specified supply voltage.

-target *target_objects*

An optional list of target nets, transistor gate pins, or output ports. If target objects are specified, only the variation values for the trigger-target pairs are reported.

trigger_objects

The list of trigger nets or transistor gate pins reported by the command. This list defines a set of delay calculation trigger pins or nets.

DESCRIPTION

The **report_pin_variation** command reports variation values that override the variation factors calculated by NanoTime at specified locations in the design.

In the command, you may restrict the types of timing arcs which will be reported by the command (min, max, rise, fall) and a list of objects that define the delay arcs which will be reported. By default all types (min,max,rise,fal) will be reported. You specify the reported delay arcs with a list of trigger objects as an argument to the command. An optional list of target objects can be used with the **-target** option to be more specific.

COMMAND PHASING

The **report_pin_variation** command can be executed any time after the **link_design** command.

SEE ALSO

```
set_pin_variation(2)
remove_pin_variation(2)
```

report_port

Generates a report on the ports in the design.

SYNTAX

```
status report_port
   [-verbose]
   [-drive]
   [-input_delay]
   [-output_delay]
   [-nosplit]
   [port_list]
```

Data Types

```
   port_list         list
```

ARGUMENTS

-verbose

Causes the report to include all port information. Otherwise, a summary report is generated that lists the ports, directions, capacitance, and attributes.

-drive

Reports drive resistance, input transition time, and driving cell information for input and inout ports.

-input_delay

Reports port input delays set with the **set_input_delay** command.

-output_delay

Reports port output delays set with the **set_output_delay** command.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

port_list

A list of ports to report. Each element in this list is either a collection of ports or a pattern matching the port names. Without this option, all ports are reported.

DESCRIPTION

This command displays information about ports in the current design. The types of port information reported by the command depend on the options used. By default, the command produces a brief report on all ports in the design.

Using the **-input_delay** or **-output_delay** option reports the minimum rise, minimum fall, maximum rise, maximum fall delays, and the clock to which the delay is relative. If "(f)" follows the clock name, the delay is relative to the falling edge of the clock; otherwise the delay is relative to the rising edge. If "(l)" follows the clock name, input delay is considered to be coming from a level-sensitive latch, or output delay is considered to be going to a level-sensitive latch. By default, the delay is relative to a rising-edge-triggered device.

COMMAND PHASING

The **report_port** command can be executed at any time after the **link_design** command.

EXAMPLES

This example shows the default port report.

```
nt_shell> report_port

 ...
Attributes:
    s  - Same cycle checking
    n  - Next cycle checking

                  Min Rise Min Fall Max Rise Max Fall
Port          Dir         Cap      Cap      Cap      Cap Attrs
-------------------------------------------------------------
clk1          inout      0.00     0.00     0.00     0.00
clk2          inout      0.00     0.00     0.00     0.00
in0           inout      0.00     0.00     0.00     0.00
out1          inout      0.00     0.00     0.00     0.00     s
out2          inout      0.00     0.00     0.00     0.00     s
```

SEE ALSO

```
create_port(2)
get_ports(2)
set_input_delay(2)
set_input_transition(2)
set_output_delay(2)
```

report_rail_net_resistance

Reports rail net resistances for nets or pins

SYNTAX

```
report_rail_net_resistance
   [net_or_pins]
```

Data Types

```
[nets_or_pins]       names or collection of net or pin objects
```

ARGUMENTS

net_or_pins

Limits the report to the specified nets or pins.

DESCRIPTION

The **report_rail_net_resistance** command generates a report on the rail net resistances computed when **parasitics_enable_rail_net_resistance** is set to true.

With no arguments, **report_rail_net_resistance** reports the number of ideal source and load pins identified for each rail net.

If nets are specified, the source and load pin report is limited to just those rail nets.

If pins are specified, the resistor network size and equivalent resistance are reported for each pin. The equivalent resistance is computed from the pin to all ideal source nodes. This equivalent resistance is for reporting purposes only, as delay calculation uses the full network that supplies current to devices in the stage.

COMMAND PHASING

The **report_rail_net_resistance** command can be executed at any time after the **link_design** command.

EXAMPLES

Example with no arguments:

```
nt_shell> report_rail_net_resistance

  Ideal
Source Pin Device Pin
  Count      Count    Net
---------- ---------- ----------
    1          55     vdd_0
    1          52     vdd_1
    1          55     vdd_2
    --         --     vnw
    --         --     vss
```

Example with nets specified:

```
nt_shell> report_rail_net_resistance vdd_0
  Ideal
Source Pin Device Pin
  Count      Count    Net
---------- ---------- ----------
    1          55     vdd_0
```

Example with pins specified:

```
nt_shell> report_rail_net_resistance [get_pins -leaf -of vdd_0]

            Equivalent
 Resistor   Resistance
  Count       (ohms)    Pin
---------- ------------ ----------
    21          605.58  xi4/inv/mp0/main/s
    24          485.52  xi2/inv/mp0@4/main/s
    24          485.52  xi2/inv/mp0@3/main/s
    22          496.63  xi2/inv/mp0@2/main/s
    21          467.24  xi0/inv/mp0/main/s
    21          678.28  xi5/inv/mp0/main/s
    95          641.76  xi3/inv/mp0@47/main/s
    95          641.76  xi3/inv/mp0@46/main/s
```

SEE ALSO

parasitics_enable_rail_net_resistance(3)

report_si_convergence

Reports crosstalk analysis convergence information for the most recently completed iteration of path tracing.

SYNTAX

```
status report_si_convergence
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command reports crosstalk analysis convergence information for the most recently completed iteration of path tracing. It can be used only after one or more path-tracing iterations have been run, and only when crosstalk analysis is enabled.

The crosstalk analysis convergence information includes the number of coupled nets, number of "effective" coupled nets (where the cross-coupling capacitance has not been filtered out) the current path-tracing iteration number, and the number and percentage of cross-coupled nets reevaluated in the current iteration.

This command is executed implicitly upon completion of each analysis iteration. You can use the command explicitly to get the convergence report again (for example, after you use the **report_paths** command).

COMMAND PHASING

The **report_si_convergence** command can only be executed after path tracing is complete.

EXAMPLES

The following command generates a crosstalk analysis convergence report.

```
nt_shell> report_si_convergence
Reporting SI convergence criteria
---------------------------------
  Total number of nets   : 21
  Number of coupled nets : 6
  Effective nets         : 5

  current iteration      : 3
  iteration limit        : 3

  Reevaluated  nets      : 3
  reevaluated net limit  : 0
  reevaluated net pct : 14.286
  reeval net pct limit: 0.000
  coupled reevaluated net pct      : 50.000
  coupled reevaluated net pct limit: 0.000

  max delta delay      : 0.190
  max delta delay limit: 0.000
  min delta delay      : -0.190
  min delta delay limit: 0.000
```

SEE ALSO

```
report_paths(2)
trace_paths(2)
si_enable_analysis(3)
```

report_si_delay_analysis

Reports the reselection parameters previously set on nets with the **set_si_delay_analysis** command.

SYNTAX

```
status report_si_delay_analysis
   [-reselected]
   [-ignored_arrival]
   [-excluded]
   [-nosplit]
   [nets]
```

Data Types

```
   nets          list
```

ARGUMENTS

-reselected

Reports the nets chosen for reselection with the **-reselect** option of the **set_si_delay_analysis** command.

-ignored_arrival

Reports the nets chosen for infinite arrival analysis using the **-ignore_arrival** option of the **set_si_delay_analysis** command.

-excluded

Reports the nets chosen to be excluded from reselection with the **-exclude** option of the **set_si_delay_analysis** command.

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

nets

A list of nets considered for the report. In the absence of this option, the command considers all nets.

DESCRIPTION

This command reports nets that have been previously chosen for reselection, exclusion from reselection, and infinite arrival analysis using the **set_si_delay_analysis** command.

Use the **-reselected**, **-excluded**, or **-ignored_arrival** options to reduce the scope of the report. Otherwise, all types of settings made with the **set_si_delay_analysis** command are reported.

Specify a list of nets for reporting to reduce the scope of the report. Otherwise, the command considers all nets in the design.

COMMAND PHASING

The **report_si_delay_analysis** command provides meaningful results only after one or more **set_si_delay_analysis** commands have been executed.

EXAMPLES

The following example excludes certain nets from reselection as victims and aggressors, and then reports the settings.

```
nt_shell> set_si_delay_analysis -exclude \
          -victims [get_nets CLK_NET*]

nt_shell> set_si_delay_analysis -exclude \
          -aggressors [get_nets AGG_NET*]

nt_shell> set_si_delay_analysis -exclude \
          -victims [get_nets V_NET] \
          -aggressors [get_nets A_NET]

nt_shell> report_si_delay_analysis -excluded
 ...
Victim net   Aggressor net   Delay Analysis attributes
-----------------------------------------------------
CLK_NET1     *               E{ mr mf Mr Mf }
CLK_NET2     *               E{ mr mf Mr Mf }
CLK_NET3     *               E{ mr mf Mr Mf }
*             AGG_NET1       E{ mr mf Mr Mf }
*             AGG_NET2       E{ mr mf Mr Mf }
V_NET         A_NET          E{ mr mf Mr Mf }
```

In the report, the asterisk character (*) indicates all nets. For example, CLK_NET1 is excluded as a victim net when coupled with any other net. The following letter codes describe the settings: E means excluded, m means minimum-delay analysis, M means Maximum-delay analysis, r means rising transitions, and f

means falling transitions.

SEE ALSO

remove_si_delay_analysis(2)
report_si_nets(2)
set_si_delay_analysis(2)
si_enable_analysis(3)

report_si_nets

Reports nets that have crosstalk delay data according to the worst delta delays or delta delay ratios found in the design.

SYNTAX

```
status report_si_nets
   [-cost_type net_cost_type]
   [-min_active_aggressors min_agg]
   [-max_nets m_nets]
   [-related_nets]
   [-min]
   [-max]
   [-transition_time]
   [-significant_digits digits]
   [-nosplit]
   [nets]
```

Data Types

net_cost_type	string
min_agg	integer
m_nets	integer
digits	integer
nets	list

ARGUMENTS

-cost_type *net_cost_type*

The cost type used for sorting the reported nets. Valid values are **delta_delay** or **delta_delay_ratio**. The value **delta_delay** causes the nets to be reported in order of decreasing absolute delta delay, whereas a value of **delta_delay_ratio** causes the nets to be reported in order of decreasing relative delta delay (absolute delta delay divided by stage delay).

-min_active_aggressors *min_agg*

The minimum number of active aggressors for a net to be reported. By default, a net is reported if affected by any number of aggressors. If you use this option, a column is added to the report showing the number of aggressors acting on each net.

-max_nets *m_nets*

The maximum number of nets to be reported. The default is 10.

-related_nets

Reports the nets related to each net affected by crosstalk. If you use this option, a column is added to the report showing the nets that are related to each listed victim net. A net is related to a victim net if it is connected to that net through one or more source-drain channels of conducting transistors. Related nets reported in the table show the same delta delay numbers (Max Delta Delay, Max Ratio, Min Delta Delay, and so on).

-min

Reports data used during minimum-delay analysis only.

-max

Reports data used during maximum-delay analysis only.

-transition_time

Reports the minimum and maximum full transition time values found at the listed nets.

-significant_digits ***digits***

Specifies the number of digits to the right of the decimal point shown in values reported by the command. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

nets

A list of nets that can be reported. If no list is specified, all nets with delta delay are reported.

DESCRIPTION

This command reports the nets that have crosstalk delta delay according to the worst delta delays or delta delay ratios found in the design. It can be used only after the **trace_paths** command has been run, and only when crosstalk analysis is enabled.

The cost function set with the **-cost_type** option determines the order in which the nets are reported:

delta_delay

The cost factor is calculated by determining the worst delta delay on the victim net. This is useful for finding the largest delta delays in the design that contribute to failing paths.

delta_delay_ratio

The cost factor is calculated by determining the worst delta delay ratio (ratio of delta delay to total stage

delay) on the victim net. This cost factor is useful for finding the delta delays that result in the largest percentage change of stage delay in a failing path.

The reports for minimum-delay analysis and maximum-delay analysis can be different. By default, both are reported. Use the **-max** or **-min** option to restrict the scope of reporting.

You should investigate cases where a large number of aggressors affect a single victim net, because the implementation tool might not be crosstalk-aware, or you might not be taking advantage of the fact that the chance of many aggressors switching at the same time is very low.

To report only the victim nets that have more than a certain number of active aggressors, use the **-min_active_aggressors** option. The number of active aggressors is defined as the number of effective aggressors minus the number of completely quiet aggressors for a given situation. If neither the **-min** nor **-max** option is specified, a completely quiet aggressor must be quiet for all four situations: min rise, min fall, max rise, and max fall. For maximum-delay reporting, a completely quiet aggressor must be quiet for max rise and max fall.

COMMAND PHASING

The **report_si_nets** command can be executed at any time.

EXAMPLES

The following command reports all nets that have crosstalk delta delay for maximum-delay analysis, listed in order of decreasing delta delay, for the first five nets with the largest delta delays.

```
nt_shell> report_si_nets -cost_type delta_delay -max_nets 5 -max
 ...
Max Delta           Min Delta
  Delay   Max Ratio   Delay   Min Ratio netname
--------- --------- --------- --------- ---------------------
    0.596     0.874    -0.373    -0.716 S[9]
    0.539     0.762    -0.344    -0.706 S[22]
    0.478     0.814    -0.352    -0.682 S[46]
    0.462     0.964    -0.290    -0.694 S[0]
    0.459     0.830    -0.339    -0.686 S[12]
```

SEE ALSO

report_paths(2)
si_enable_analysis(3)

report_si_noise_analysis

Reports the exceptions previously set on nets with the **set_si_noise_analysis** command.

SYNTAX

```
status report_si_noise_analysis
   [-exclude]
   [-nosplit]
   [nets]
```

Data Types

nets	list

ARGUMENTS

-exclude

Reports the nets chosen to be excluded from noise analysis with the **-exclude** option of the **set_si_noise_analysis** command.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

nets

A list of nets considered for the report. In the absence of this option, the command considers all nets.

DESCRIPTION

This command reports the nets that have been previously chosen for exclusion from noise analysis using the **set_si_noise_analysis** command.

You can use the **-exclude** option to reduce the scope of the report. Otherwise, all types of settings made with the **set_si_noise_analysis** command are reported.

If you specify a list of nets, then only those nets are considered for reporting. Otherwise, the command considers all nets in the design.

COMMAND PHASING

The **report_si_noise_analysis** command can be executed any time after the **check_design** command.

EXAMPLES

The following example excludes certain nets from noise analysis as victims and aggressors, and then reports the settings.

```
nt_shell> set_si_noise_analysis -exclude \
          -victims [get_nets CLK_NET*]

nt_shell> set_si_noise_analysis -exclude \
          -aggressors [get_nets AGG_NET*]

nt_shell> set_si_noise_analysis -exclude \
          -victims [get_nets V_NET] \
          -aggressors [get_nets A_NET]

nt_shell> report_si_noise_analysis -exclude
 ...
Victim Net  Aggressor Net  Status    Analysis Type
----------- -------------- --------- ---------------------------------------
CLK_NET1      *            Excluded  Above_High Below_High Above_Low Below_Low
CLK_NET2      *            Excluded  Above_High Below_High Above_Low Below_Low
CLK_NET3      *            Excluded  Above_High Below_High Above_Low Below_Low
   *         AGG_NET1      Excluded  Above_High Below_High Above_Low Below_Low
   *         AGG_NET2      Excluded  Above_High Below_High Above_Low Below_Low
V_NET        A_NET         Excluded  Above_High Below_High Above_Low Below_Low
```

In the report, the asterisk character (*) indicates all nets. For example, CLK_NET1 is excluded as a victim net when coupled with any other net.

SEE ALSO

```
remove_si_noise_analysis(2)
report_noise(2)
set_si_noise_analysis(2)
si_enable_noise_analysis(3)
```

report_simulation

Reports information about a simulation object or generates a SPICE deck or vector template.

SYNTAX

```
status report_simulation
   [-verbose]
   [-spice_deck]
   [-vectors]
   [-template]
   [-nosplit]
   sim_object_list
```

Data Types

```
   sim_object_list     collection
```

ARGUMENTS

-verbose

Generates a detailed report showing the names as well as the numbers of inputs, nets, and transistors, and other information about the simulation region.

-spice_deck

Writes out the simulation region as a SPICE deck, which you can redirect into a file.

-vectors

Writes out the simulation region vectors, which you can redirect into a file.

-template

Writes out a vector-file template for the simulation region, which you can redirect into a file and then edit to specify the vectors for simulation.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

sim_object_list

A list of one or more simulation objects collected with the **all_simulations** or **get_simulations** commands.

DESCRIPTION

The **report_simulation** command reports the inputs, outputs, nets, and transistors of a specified simulation region. It can also generate a SPICE deck or a vector template file for a simulation region.

The **report_simulation** command is used after the **check_design** command, which creates the simulation regions. The simulation objects must be collected with the **all_simulations** or **get_simulations** command, typically by setting a variable equal to the collection.

The **-spice_deck** option writes out the region as a SPICE deck, which you can redirect into a file and then use for simulation externally.

The **-template** option writes out a vector-file template for the region, which you can redirect into a file and then edit to specify the vectors for simulation. The **-vectors** option writes out the current set of vectors for the region for you to examine.

COMMAND PHASING

The **report_simulation** command must be executed after the **check_design** command but before the **trace_paths** command.

EXAMPLES

The following example generates a collection containing the simulation region that has dec2_n as an output, and then generates a verbose report on the region's inputs, outputs, nets, and transistors:

```
nt_shell> set mysim [get_simulations -input in1]
 ...
nt_shell> report_simulation -verbose $mysim
 ...
>>>
nt_shell> report_simulation -verbose \
          [get_simulations -output dec2_n]
get_simulations: outputNetlist length: 1
get_simulations: blockList length: 1

Detailed Simulations Report

Attribute       Value
--------------- ------------------------------
Output Nets     dec2_n
Input Nets      {ip1 ip0_n}
Internal Nets   u5.N1N6
```

```
Transistors     {u5.MM2 u5.MM1 u5.MM3 u5.MM4}
Spice Header    /remote/Data/header.sp
Control Header  /remote/Data/control_header.h
Vectors
```

The following command generates a SPICE deck for the region stored in the variable mysim and redirects the output to a file called myspice.spi:

```
nt_shell> report_simulation -spice_deck \
          $mysim > myspice.spi
 ...
```

The following command generates a vector template for the region containing dec2_n as an output:

```
nt_shell> report_simulation -template \
          [get_simulations -output dec2_n]
get_simulations: outputNetlist length: 1
get_simulations: blockList length: 1

;
-----------------------------------------------------

Outputs:
dec2_n

Signals:
ip1  ip0_n

Vectors:
; ip1_VALUE (DELAY_VALUE)  ip0_n_VALUE (DELAY_VALUE)
;  1 (0.0)  1 (0.0)

; States:
; net1 net2 ...

; Initial Conditions:
; v1 v2 ...

Options:
min max


;
-----------------------------------------------------
```

To invoke the edited vector file for the simulation region, use the **set_simulation_attributes** command.

You can redirect the vector template to a file and edit it to create a new vector file. You can then apply the new file to the simulation region with the **set_simulation_attributes -vectors** command.

SEE ALSO

```
all_simulations(2)
check_design(2)
```

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

report_storage_node

Reports storage nodes in the current design.

SYNTAX

```
status report_storage_node
   [-verbose]
   [-errors error_type]
   [-nosplit]
   [nets]
```

Data Types

```
   error_type     string
   nets           collection
```

ARGUMENTS

-verbose

Displays detailed information about the storage node.

-errors *error_type*

Displays the number of storage nodes in error conditions. The *error_type* argument accepts three values: **not_marked** (storage nodes which are not marked or recognized as any sequential topology), **not_clocked** (a clock does not propagate to the channel-connected block containing the storage node), and **all** (includes both errors).

-nosplit

Prevents displaying text that spans multiple lines in the report. By default, if the text in a column exceeds the column width, the line is split and next column begins on the next line. Using the **-nosplit** option produces long and unbroken lines.

nets

The list or collection of nets to consider for reporting.

DESCRIPTION

The **report_storage_node** command reports storage nodes in the current design. If used without any options, the command displays the summary report. The summary includes the counts of valid sequential topologies, invalid sequential topologies and storage nodes which are not marked or recognized as any sequential topologies. The invalid sequential topologies are user-defined or automatically recognized latches, precharges, or register files which are not driven by clocks.

Use the **-verbose** option to get detailed information about the storage nodes. Use the **-errors** option to get the error information. Limit the storage node reporting by specifying a collection of nets to evaluate.

COMMAND PHASING

The **report_storage_node** command provides meaningful results only after one or more **match_topology** commands have been executed.

EXAMPLES

The following example shows the summary report of storage nodes in the current design.

```
nt_shell> report_storage_node

****************************************
Report : storage node
Design : top
****************************************

Type                              Count
------------------------------ --------
Valid Sequential Marking             11
Invalid Sequential Marking            0
No Marking                            6
----------------------------------------
Total                                17
```

SEE ALSO

remove_storage_node(2)
report_channel_connected_block(2)
report_topology(2)

report_technology

Reports the details of transistor models available in the in-memory technologies.

SYNTAX

```
status report_technology
   [-model model_name]
   [-name tech_name]
   [-voltage volt]
   [-temperature temp]
   [-index]
   [-ranges]
   [-soi_parameters]
```

Data Types

```
   model_name       string
   tech_name        string
   volt             float
   temp             float
```

ARGUMENTS

-model *model_name*

The name of the transistor model to report. If this option is not used, all models are reported.

-name *tech_name*

Restricts reporting to the specified technology. If this option is not used, all technologies are reported.

-voltage *volt*

Restricts reporting to the specified operating voltage. If this option is not used, all operating voltages are reported.

-temperature *temp*

Restricts reporting to the specified operating temperature, in degrees Celsius. If this option is not used, all operating temperatures are reported.

-index

Causes the report to show index values (different parameter settings) supported by the technology.

-ranges

Causes the report to show the range (minimum and maximum) of index values supported by the technology.

-soi_parameters

Causes the report to show the SOI parameters, both user-specified and default, for all the models to be generated.

DESCRIPTION

The **report_technology** command reports the details of transistor model entries in the "in-memory technologies," which are the technologies used in the NanoTime timing analysis.

If a transistor model is present in an input netlist but is never referenced by a device, that model is not part of the in-memory technologies. Similarly, if a model is initially associated with a device, but the model is later replaced due to back-annotation of the device with a DSPF parasitics file, the original model is no longer part of the in-memory technologies. Models that are not part of the in-memory technologies are not reported by the **report_technology** command.

Technology models can be read in explicitly with the **read_spice_model** command or **read_techfile** command, or read implicitly with the **link_design** command.

Here is an example of a technology report:

```
nt_shell> report_technology
Model Tech.
Name  Name  Volt  Temp   Width   Length   Delvto  Mulu0  Total
----- ----- ----- ------ ------- -------- ------- ------ -----
nch   typ   1.20  85.00  2       0.100um  0.000   1.000  2
pch   typ   1.20  85.00  5       2        0.000   1.000  5
1
```

For each transistor model, the report shows the following information:

- Transistor model name
- Technology name
- Voltage (volts)
- Temperature (degrees Celsius)
- Width of transistor channel or number of different widths
- Length of transistor channel or number of different lengths
- Delta threshold voltage offset, 0.0 by default
- Electron mobility adjustment multiplier, 1.0 by default

- Total number of transistor models

The columns Width, Length, Delvto, and Mulu0 show the number of index values supported by the transistor model, or the value itself if there is only a single value in the transistor model. The total number of transistor models is the number of supported combinations of Width, Length, Delvto, and Mulu0.

By default, transistor models in all technologies are reported. To restrict the scope of the report to specific transistor model name, technology name, voltage, or temperature, use the **-model**, **-name**, **-voltage**, or **-temperature** option. To show the range of index values (minimum and maximum) for Width, Length, Delvto, and Mulu0, use the **-ranges** option. Use the **-index** option to list all index values.

To get a listing of technology data sources, use the **list_technology** command.

COMMAND PHASING

The **report_technology** command should be issued after the **check_design** command.

EXAMPLES

The following command reports the "pch" transistor model in the "typ" technology and shows the width and length index values for the model.

```
nt_shell> report_technology -model pch -name typ -index

Model  Tech.
Name   Name   Volt  Temp   Width  Length  Delvto  Mulu0  Total
------ ------ ----- ------ ------ ------- ------- ------ -----
pch    typ    1.00  85.00  5      2       0.000   1.000  5

Length index list:
        0.200um
Width index list:
        0.200um

Length index list:
        0.100um
Width index list:
        4.444um      2.222um      1.111um      0.600um
```

SEE ALSO

```
link_design(2)
list_technology(2)
read_spice_model(2)
register_netlist(2)
set_technology(2)
```

report_topology

Reports the current state of topology recognition.

SYNTAX

```
status report_topology
   [-structure_type type]
   [-verbose]
   [-nosplit]
   [topology_list]
```

Data Types

```
   type              string
   topology_list     list
```

ARGUMENTS

-structure_type *type*

Restricts topology report to only those of the specified structure type. The allowed values for structure type are **clock_gate**, **cross_coupled**, **cross_coupled_pmos**, **differential_synchronizer**, **feedback**, **flip_flop**, **inverter**, **latch**, **memory_precharge**, **memory_write_circuit**, **mux**, **nand**, **nor**, **power_switch**, **precharge**, **pulldown**, **pullup**, **ram**, **register**, **sense_amp**, **tgate**, **turnoff**, **weak_pullup**, and **xor**.

-verbose

Reports details about the topology structures rather than just the number of structures.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and the next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

topology_list

A collection of topologies in which to search for the specified structure (for example, a collection obtained by **get_topology**).

DESCRIPTION

The **report_topology** command reports the topology structures in the design, whether the structures were recognized automatically by NanoTime or marked manually with the **mark_*** commands (**mark_feedback**, **mark_latch**, and so on).

The "pending" state means the topology of those types of structures are not recognized yet. This occurs when the **report_topology** command is run before the **match_topology** command. After the **match_topology** command, the term "pending" appears for structure types that are not automatically recognized.

By default, the command lists the structure types and reports the number of occurrences of each type of structure in the design. For information about each structure, use the **-verbose** option. To restrict the report to just one type of structure, use the **-structure_type** option and specify the structure type.

COMMAND PHASING

The **report_topology** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command reports the number of instances of each type of topology recognized or marked in the design.

```
nt_shell> report_topology
 ...
                                      User   Global
                   Auto Manually  Library  Library
Structure Type    Marked   Marked   Marked   Marked    Total
--------------- -------- -------- -------- -------- --------
clock_gate           192        0        0        0      192
cross_coupled          0        0        0        0        0
cross_coupled_pmos     0        1        0        0        1
feedback             224        0        0        0      224
inverter            2723        0        0        0     2723
latch                128        0      128       64      320
flip_flop       disabled        0        0        0        0
 ...
```

The following command reports the location of each feedback structure recognized or marked in the design.

```
nt_shell> report_topology -structure_type feedback -verbose
 ...

Attributes:
    u - User Defined
```

```
    C - Common Topology Library Defined
    U - User Topology Library Defined
    G - Global Topology Library Defined

Feedback                          Attrs
--------------------------------- -----
Xaddsub.Xadder.Xla2.Xlgen.XC3.Mp1
Xaddsub.Xadder.Xla5.Xlgen.XC4.Mp1
Xaddsub.Xadder.Xla8.Xlgen.XP4.Mp1
Xaddsub.Xadder.Xla5.XG0.X0.Mp1
Xaddsub.Xadder.Xla14.XG0.X0.Mp1
Xaddsub.Xadder.Xla15.Xlgen.XC2.Mp1
 ...
```

The following command reports the location of each latch structure recognized or marked in the cell called Xaddsub.

```
nt_shell> report_topology -structure_type latch \
          -verbose [get_topology -of_objects [get_cells Xaddsub]]
```

SEE ALSO

```
get_lib_topology(2)
get_topology(2)
list_lib_topology(2)
report_topology_library(2)
```

report_topology_library

Displays a report on lib_topology objects in the design.

SYNTAX

```
status report_topology_library
   [-enabled]
   [-only_used]
   [lib_topology_list]
```

Data Types

```
   lib_topology_list     list
```

ARGUMENTS

-enabled

Restricts the scope of the report to the library topologies currently enabled for searching.

-only_used

Restricts the scope of the report to the library topologies found to exist in the design by the **set_search_enabled -only_used** command.

lib_topology_list

Restricts the scope of the report to the listed lib_topology objects.

If you specify a list of lib_topology objects, only those in the list are reported. Each lib_topology must be specified in the following form:

library_name.pattern

where *library_name* is the name of the topology library, the period is the hierarchy separator character (which can be changed by setting the **hierarchy_separator** variable), and *pattern* is the lib_topology name or a wildcard pattern that matches one or more lib_topology names.

DESCRIPTION

This command displays a report on lib_topology objects in the design. The report shows the lib_topology name, search type (either "pattern" or the subcircuit name), the number of occurrences matched in the design, the number of occurrences marked in the design, the enable status (either enabled or disabled for search), and types of action taken (before match_topology, after clock propagation, and so on).

The **report_topology_library** command without any options reports on all lib_topology objects. The command options restrict the scope of the report. The **-enable** option reports only the library topologies currently enabled for searching (for example, by the **set_search_enabled** command). The **-only_used** option reports only the lib_topology objects found to exist in the design, based on the value of the match_count attribute of the lib_topology objects.

Note: **Matched** means how many occurrences of the pattern are matched in the design. **Marked** means how many topologies are marked by the matched pattern. When a pattern is found, NanoTime tries to execute the actions in the .libt file. Sometimes the .libt file will cause checking of the clock or some other condition before executing the **mark_*** command. If the condition is not satisfied, the pattern is matched, but the topology is not marked.

Command Phasing

The **report_topology_library** command can be executed in any state after "netlist-linked" up to and including "paths-traced".

EXAMPLES

The following command reports the lib_topology named nando_istack2 in the common library.

```
nt_shell> report_topology_library common.nando_istack2
 ...
Attributes:
    + - Enabled for search
    - - Disabled for search
    m - Pre match topology action (unclocked)
    c - Post clock propagation action
    M - Post match topology action (clocked)
    t - Pre check topology action
    T - Post check topology action
    d - Pre check design action
    D - Post check design action

Library name: common
Total number of lib_topologies in library: 24

Name           Search type ...  Matched   Marked   Attribute
-------------- ---------------- --------- -------- ---------
nando_istack2  pattern          128       128      +c
```

The following command reports the lib_topology objects in the common library that exist in the design.

```
nt_shell> report_topology_library -only_used common.*
```

```
...
```

The following command reports the lib_topology objects in the common library that are enabled for searching.

```
nt_shell> report_topology_library -enabled common.*
 ...
```

SEE ALSO

```
get_lib_topology(2)
list_lib_topology(2)
match_topology(2)
read_topology_library(2)
set_search_enabled(2)
hierarchy_separator(3)
```

report_transistor_direction

Reports transistor signal flow direction.

SYNTAX

```
status report_transistor_direction
   [-bidirectional]
   [-nondirectional]
   [-transistor cell_list]
   [-channel_connected_block]
   [-nosplit]
   [-unset]
```

Data Types

```
   cell_list     list
```

ARGUMENTS

-bidirectional

Lists bidirectional transistors, including transistors whose directions were determined by the tool and transistors whose directions were manually specified.

-nondirectional

Lists nondirectional transistors, including transistors whose directions were determined by the tool and transistors whose directions were manually specified.

-transistor

Reports the direction of the transistors included in the *cell_list*.

-channel_connected_block

Must be used with the **-transistor** option. Reports the SPICE netlists of the channel-connected blocks of the transistors specified in the *cell_list*.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces

long, unbroken lines instead.

-unset

Reports bidirectional and nondirectional transistors not set by the user.

DESCRIPTION

The **report_transistor_direction** command, when used without any options, provides a summary report of the number of transistors that are bidirectional, the number of transistors with a direction automatically set by the **match_topology** or **check_topology** commands, and the number of transistors with a direction set by the **set_transistor_direction** command. This information can be helpful in determining the success of topology recognition.

Thee **-bidirectional** option of the **report_transistor_direction** command generates a list of all of the bidirectional transistors. The **-nondirectional** option generates a list of all the nondirectional transistors.

The **-transistor** option generates a list of all the transistor directions of the specified cells. The **-channel_connected_block** option, when used in addition to the **-transistor** option, provides the SPICE netlists of the channel-connected blocks of the specified cells.

COMMAND PHASING

The **report_transistor_direction** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command generates a summary report of transistor direction.

```
nt_shell> report_transistor_direction
 ...
Type                                    Count
---------------------------------- ----------
Bidirectional (Unset)                     300
Bidirectional (User Defined)               10
Nondirectional                             10
Unidirectional (Automatically Set)      10584
Unidirectional (User Defined)               0
-------------------------------------------
Total                                   10904
```

The following command lists all the bidirectional transistors.

```
nt_shell> report_transistor_direction -bidirectional
 ...
Bidirectional Transistor
```

```
    -------------------------------------------
    Xaddsub.Xadder.Xfa0.Mn4
    Xaddsub.Xadder.Xfa0.Mn5
    Xaddsub.Xadder.Xfa0.Mn6
    Xaddsub.Xadder.Xfa0.Mn7
     ...
```

SEE ALSO

check_topology(2)
match_topology(2)
set_transistor_direction(2)

report_variation

Reports user-defined random on-chip-variation design information.

SYNTAX

```
status report_variation
   [-nosplit]
   [-verbose]
   [-list_unmatched]
   [-max_unmatched num]
   [-show_wrapper_parameters]
```

ARGUMENTS

-nosplit

Prevents the splitting of lines in the report. Information in the report appears in fixed-width columns. By default, if the information in a given field exceeds the column width, the next field begins on a new line, starting in the correct column. The **-nosplit** option prevents line splitting, resulting in long, unbroken lines in the report.

-verbose

This option only applies when timing_enable_slew_variation is set to TRUE. For that scenario without the -verbose option set, the first delay variation value only is displayed with an asterisk next to it, indicating more data is present. When the -verbose option is used then the existing column var pct, and the new columns "input trans" and "tans var pct" will display a list of values that correspond to the values in the input trans list.

-list_unmatched

Causes the transistors that are not matched to any variation parameters to be listed. If -max_unmatched option is not used, maximum 10 transistors are listed for each min and max corner.

-max_unmatched num

Specifies the maximum number of transistors to report for the -list_unmatched option for each min and max corner.

-show_wrapper_parameters

Causes the wrapper parameters to be listed when -list_unmatched options is enabled.

DESCRIPTION

The **report_variation** command generates a report of user-defined random on-chip-variation design information. These parameters are categorized by transistors, dynamic simulation blocks, libcells, and default values. The report includes a summary of transistor matches for each transistor category defined by the user and a listing of transistors that did not match the user definitions. All transistor sizes in this report are in nanometers.

SEE ALSO

```
set_variation_parameters(2)
set_libcell_variation_parameters(2)
timing_enable_slew_variation(3)
```

report_variation_calculation

Generates a report on the local variation calculations for a path obtained with the **get_timing_paths** command.

SYNTAX

```
status report_variation_calculation
   [-significant_digits digits]
   [path_list]
```

Data Types

```
   digits          integer
   path_list       list
```

ARGUMENTS

-significant_digits *digits*

Specifies the number of digits to the right of the decimal point. In the absence of this option, the number of significant digits is determined by the **report_default_significant_digits** variable.

path_list

A list of paths generated by the **get_timing_paths** command.

DESCRIPTION

The **report_variation_calculation** command generates a report of the calculations used to compute a parameterized on-chip variation (POCV) adjustment for a slack calculation.

The **report_paths** command displays the adjustments made by POCV. To find out the size of the POCV adjustment, use the option **-path_type full** or **-path_type full_clock** in the **report_paths** command.

The **report_variation** command generates a report that shows variable settings, the variation of the launch path and the capture paths, and the nominal delays of the launch and capture paths not including the common mode path delay. Path variation values do not use the common mode portion of these path.

The **report_variation_calculation** command generates a report on the POCV calculations for one or more paths obtained with the **get_timing_paths** command, which creates a collection of paths taken from the paths stored in the path database.

You can set a variable to a collection created by the **get_timing_paths** command, and then use the **report_variation_calculation** command to get information about the paths in the collection. For example:

```
nt_shell> set path1 [get_timing_paths -max -max_paths 3]
nt_shell> report_variation_calculation $path1
```

The slack adjustment made to a path for POCV is based on the variable settings at the time the path was found. To get a variation value based on a different sigma or variation parameter the path database must be updated with a new path trace. To update the path database and path reports with new POCV variable settings, use the **reset_design -paths** command, followed by the **trace_paths -pocv** command.

Path Segments

Variables that Control POCV Analysis

NanoTime uses a set of application variables to adjust the delays for each path segment. The **timing_pocv_sigma** variable determines the number of standard deviations of variation used while computing a slack adjustment. The **timing_pocv_gate_delay_only** variable determines whether the variation adjustments exclude the wire delay portion of each stage delay.

Slack Adjustment Calculation for All Checks

```
Nominal Slack = (Required Time) - (Arrival Time) - (Clk Unc)
Slack Variation = SQRT( launch path variation**2 + capture path variation**2)
Adjusted Slack = nominal slack - slack variation
```

COMMAND PHASING

The **report_variation_calculation** command must be executed after path tracing is complete.

EXAMPLES

The following command generates a variation/POCV report for the worst-case (least-slack) hold check stored in the path database.

```
nt_shell> report_variation_calculation [get_timing_paths -min -max_paths 1]
```

The following command generates a POCV report for the worst-case (least-slack) setup check stored in

the path database that starts at port IN1 and ends at port OUT2.

```
nt_shell> report_variation_calculation [get_timing_paths \
          -from [get_ports IN1] \
          -to [get_ports OUT2] \
          -max -max_paths 1]
```

SEE ALSO

get_timing_paths(2)
trace_paths(2)
set_variation_parameters(2)
set_libcell_variation_parameters(2)
report_variation(2)
timing_pocv_sigma(3)
timing_pocv_gate_delay_only(3)
tech_pocv_match_voltage_pct(3)
tech_pocv_match_length_pct(3)

report_wordline

Generates a report on the wordline associated with a memory in the design.

SYNTAX

```
status report_wordline
   [-bitcells]
   [-nosplit]
   [net_list]
```

Data Types

```
   net_list      collection
```

ARGUMENTS

-bitcells

Reports the bitcells on the wordlines.

-nosplit

Prevents the splitting of lines in the report. By default, if the information in a column exceeds the column width, the line is split and next column begins on the next line. The **-nosplit** option produces long, unbroken lines instead.

net_list

A list of wordlines to be reported. Each element in this list is either a collection of wordline nets or a pattern matching the net names. Without this option, all wordlines are reported.

DESCRIPTION

This command displays information about wordlines in the current design. The type of wordline information reported by the command depends on the options used. By default, the command produces a list of

wordlines items and their values.

Using the **-bitcells** option adds extra rows in the items list for the bitcell name and bitline for each bit cell associated with the wordline.

COMMAND PHASING

The **report_wordline** command can be executed at any time after the **link_design** command.

EXAMPLES

This example shows the default wordline report.

```
nt_shell> report_wordline 'Xmem/rwaddr0'

 ...
Attributes:
    s - selected

Memory wordline 'Xmem/rwaddr0':

Item               Value
---------------- - -------------------------------------------------
Memory             read_mem
Num Bit Cells      2
```

This example shows the wordline report with the **-bitcells** option.

```
nt_shell> report_wordline -bitcells Xmem/bit0

 ...
Attributes:
    s - selected

Memory wordline 'Xmem/rwaddr0':

Item               Value
---------------- - -------------------------------------------------
Memory             read_mem
Num Bit Cells      2
Bitlines    0      {Xmem/bit0 Xmem/bitb0}
Bit-cell    0    s {Xmem/Xsram6t_0/Xsram0/b Xmem/Xsram6t_0/Xsram0/bb}
Bitlines    1      {Xmem/bit1 Xmem/Xmem0/bitb1}
Bit-cell    1    s {Xmem/Xsram6t_1/Xsram0/b Xmem/Xsram6t_1/Xsram0/bb}
```

SEE ALSO

```
create_memory(2)
get_memory(2)
report_memory(2)
```

```
report_bitline(2)
```

reset_design

Removes annotated and generated information from the design.

SYNTAX

```
status reset_design
   [-analysis_data]
   [-paths]
```

ARGUMENTS

-analysis_data

Resets the analysis data.

-paths

Resets the path database.

DESCRIPTION

The **reset_design** command resets the design, taking you back to an earlier point in the analysis flow. Use this command when you want to discard part or all of the analysis information.

If you want to use the same analysis data in the future, save it with the **save_analysis_data** command before you use the **reset_design** command. The saved analysis data can be restored later with the **restore_analysis_data** command, thereby saving path tracing time.

There are two types of information discarded by the **reset_design** command: analysis data and the path database. The analysis data includes attribute settings, timing constraints, back-annotated design data, and delay analysis results. The path database includes path timing information that can be reported with the **report_paths** command.

To remove only the analysis data, use the **-analysis_data** option. This takes you back to the analysis stage just before path tracing, which is useful when you want to run the **trace_paths** command again

with different tracing options. This option also removes the path database information if it exists.

To remove only the path database information, use the **-paths** option. This takes you back to a point in the analysis just before path tracing.

COMMAND PHASING

The **reset_design** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command resets the current design, thus discarding all back-annotated data, timing constraints, and analysis results.

nt_shell> **reset_design**

SEE ALSO

current_design(2)
report_paths(2)
restore_analysis_data(2)
save_analysis_data(2)
trace_paths(2)

restore_analysis_data

Restores analysis data previously saved with the **save_analysis_data** command.

SYNTAX

```
status restore_analysis_data
   dir_name
```

Data Types

```
   dir_name        string
```

ARGUMENTS

dir_name

Specifies the directory in which the analysis data was stored by the **save_analysis_data** command.

DESCRIPTION

This command restores the analysis data previously saved with the **save_analysis_data** command. Restoring the analysis data before running the **trace_paths** command can significantly reduce the runtime because NanoTime does not need to generate the same analysis data again.

COMMAND PHASING

The **restore_analysis_data** command can only be executed after the **check_design** command has been executed successfully.

EXAMPLES

The following example saves the analysis data into a directory called session1, and then restores that data in a later session.

```
nt_shell> trace_paths
 ...
nt_shell> save_analysis_data session1
1

next NanoTime session

nt_shell> link_design top
 ...
nt_shell> check_topology
 ...
nt_shell> check_design
 ...
nt_shell> restore_analysis_data session1
1
nt_shell> trace_paths
 ...
```

SEE ALSO

save_analysis_data(2)
trace_paths(2)

restore_session

Restores a NanoTime session from a directory saved by the **save_session** command.

SYNTAX

```
status restore_session dir_name
```

Data Types

dir_name	string

ARGUMENTS

dir_name

Specifies the name of a directory from which to read the session information.

DESCRIPTION

Use this command to read a directory that was written by the **save_session** command and restore a NanoTime session to the same state as the session where the **save_session** command was issued.

Ensure the version of NanoTime used to restore a session with the **restore_session** command is the same version used when saving the session with the **save_session** command. Locate the version used to save the session from the README file located in the session directory.

No current designs can be loaded to execute the **restore_session** command. Issue this command in a clean shell.

The command first tries to check out all licenses that were present at the time of the **save_session** command. If this fails, restoring a session is aborted.

The design data is restored from data in the session directory. The **restore_session** command reads the library files that are needed to restore the session.

The restore directory contains the lib_map ASCII file name. This file contains the path and leaf name for each cell library needed to restore the session. If necessary, you can edit the path names to give the **restore_session** command the updated locations of the needed cell library files. The leaf names of those files cannot be changed. The cell libraries are assumed to be the same as those used when the **save_session** command was issued.

When a saved session is restored, NanoTime application variables are restored to their saved values. Other Tcl variables are neither saved nor restored.

In the restored session, only standard reporting and custom reporting commands are available. This mainly consists of report_xxx and list_xxx commands for standard reporting, get_xxx and all_xxx commands to create collections, and the get_attribute command to access object attribute values.

COMMAND PHASING

The **restore_session** command can only be executed in the "startup-complete" state.

EXAMPLES

This example restores a NanoTime session from a directory named state1:

```
nt_shell> restore_session state1
```

SEE ALSO

save_session(2)

run_parallel

Launches parallel NanoTime instances.

SYNTAX

```
status run_parallel
   [-instance_namesname_list]
   [-files script_file_list]
   [-directories directory_list]
   [-parallel_instance_count number_of_instances]
   [-parallel_instance_timeout maximum_time]
   [-exclude_master]
```

Data Types

name_list	list
script_file_list	list
directory_list	list
number_of_instances	integer
maximum_time	float

ARGUMENTS

-instance_names *name_list*

Launches parallel instances of NanoTime, each with a unique value from the list. Every invocation has the same arguments as the parent instance.

-files *script_file_list*

Launches parallel instances of NanoTime, each with a unique Tcl script file from the list. The working directory is set to the directory containing the script.

-directories *directory_list*

Launches parallel instances of NanoTime, each with a unique Tcl script file from the list. Can be used to control the working directory when using the **-files** option.

-parallel_instance_count *number_of_instances*

Reduces the number of parallel instances. By default, the number of parallel instances is equal to the number of entries in the instance names list or the files list.

-parallel_instance_timeout *maximum_time*

Controls the maximum wall clock time allowed for a parallel instance. Default is infinity.

-exclude_master

Prevents the master from performing any of the parallel tasks. Default is for the master to perform one of the tasks.

DESCRIPTION

The **run_parallel** command launches multiple instances of NanoTime to reduce the wall clock time needed to solve a complex problem.

COMMAND PHASING

The **run_parallel** command is not phase-restricted.

SEE ALSO

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

save_analysis_data

Saves the current analysis data for faster operation of the **trace_paths** command in a future session.

SYNTAX

```
status save_analysis_data
   [-include incl_list]
   [-replace]
   dir_name
```

Data Types

```
   incl_list       list
   dir_name        string
```

ARGUMENTS

-include *incl_list*

Specifies whether or not to include additional data; allowable values are: **technology**.

-replace

Causes NanoTime to overwrite any existing data in the specified directory.

dir_name

Specifies the directory in which to store the analysis data.

DESCRIPTION

This command saves certain types of analysis data generated by the **trace_paths** command, including the data generated in preparation for simulation of timing arcs.

In a future session using the same design and the same technologies, you can use the **restore_analysis_data** command to restore the analysis data before using the **trace_paths** command.

This might significantly reduce the runtime for the **trace_paths** command because NanoTime does not need to generate the same analysis data again.

The **-replace** option causes NanoTime to overwrite any existing data in the specified directory. In the absence of this option, NanoTime reports an error if you try to save the analysis data to a directory already containing saved analysis data.

COMMAND PHASING

The **save_analysis_data** command can only be executed in the "paths-traced" state.

EXAMPLES

The following example saves the analysis data into a directory called session1. Data stored in that directory is restored and used in a future session to reduce the runtime for the **trace_paths** command.

```
nt_shell> trace_paths
 ...
nt_shell> save_analysis_data session1
1

next NanoTime session

nt_shell> link_design top
 ...
nt_shell> check_topology
 ...
nt_shell> check_design
 ...
nt_shell> restore_analysis_data session1
1
nt_shell> trace_paths
 ...
```

SEE ALSO

restore_analysis_data(2)
trace_paths(2)

save_session

Saves data of a NanoTime session in the named directory so that it can be restored later using the **restore_session** command.

SYNTAX

```
status save_session
   [-replace]
   dir_name
```

Data Types

```
   dir_name       string
```

ARGUMENTS

-replace

Indicates that if any file or directory with the same name as the one specified already exists in the file system, it is overwritten with the saved data for the current session. Use the **-replace** option with great caution because it can remove all the existing data, including all files and subdirectories that are already present in the target directory.

dir_name

Specifies the name of a directory in which to save the session. If the named directory does not exist, the **save_session** command tries to create it. If it already exists, the **save_session** command issues an error message and stops, unless you have specified the **-replace** option.

DESCRIPTION

This command creates a repository of data to save the current NanoTime session. The name of the directory in which to save the session is a required argument. If the directory does not exist, a new one will be created and the data for the session will be written into the directory.

If the directory exists, an error is issued unless you have set the **-replace** option. The **-replace** option causes an existing file or directory to be removed and replaced by the data of the current session to be saved.

Note: There are a few things that are not saved. These include: Tcl variables that are not application variables, collections, Tcl procedures, Tcl command history, and warning and informational message suppressions.

Ensure the version of NanoTime used to restore a session with the **restore_session** command is the same version used when saving the session with the **save_session** command. Locate the version used to save the session from the README file located in the session directory.

COMMAND PHASING

The **save_session** command can only be executed in the "paths-traced" state.

EXAMPLES

This example saves a NanoTime session in a directory named state1:

```
nt_shell> save_session state1
```

SEE ALSO

restore_session(2)

set_allow_pin_swap

Allows pins to be swapped in back annotated parasitics.

SYNTAX

```
status set_allow_pin_swap
   -cell cell_name
   -pins pin_list
```

Data Types

```
   cell        string
   pin_list       list
```

ARGUMENTS

cell_name

The name of the cell or primitive which is allowed to have swapped pins in back annotated parasitics.

pin_list

A list of exactly 2 pins that may be swapped in back annotated parasitics.

DESCRIPTION

The **set_allow_pin_swap** command declares that the named cell is allowed to have the two named pins swapped in back annotated parasitics. This may occur if the LVS (layout versus schematic) tool considers the pins to be electrically equivalent, for example the 2 terminals of a resistor subckt.

To undo the effect of the **set_allow_pin_swap** command, use **remove_allow_pin_swap**.

To report currently allowed pin swaps, use **report_allow_pin_swap**.

To globally allow pin swaps on transistor devices, use the **parasitics_enable_drain_source_swap** variable.

COMMAND PHASING

The **set_allow_pin_swap** command is used after the **link_design** command and before the **check_design** command. To be effective, it must be issued prior to the **read_parasitics** command.

RESTRICTIONS

The **set_allow_pin_swap** command does not enable swapping of pins which may be considered logically equivalent, for example 2 pins of a nand-gate.

The **set_allow_pin_swap** command may only be applied to a single pair of pins for any given cell.

EXAMPLES

The following command defines the two pins t1 and t2 of the rcdel cell to be swappable.

```
nt_shell> set_allow_pin_swap -cell rcdel -pins {t1 t2}
```

SEE ALSO

```
report_allow_pin_swap(2)
remove_allow_pin_swap(2)
parasitics_enable_drain_source_swap(3)
```

set_app_var

Sets the value of an application variable.

SYNTAX

```
string set_app_var
   -default
   var
   value
```

Data Types

```
   var        string
   value      string
```

ARGUMENTS

-default

Resets the variable to its default value.

var

Specifies the application variable to set.

value

Specifies the value to which the variable is to be set.

DESCRIPTION

The **set_app_var** command sets the specified application variable. This command sets the variable to its default value or to a new value you specify.

This command returns the new value of the variable if setting the variable was successful. If the application variable could not be set, then an error is returned indicating the reason for the failure.

Reasons for failure include:

- The specified variable name is not an application variable, unless the application variable **sh_allow_tcl_with_set_app_var** is set to true See the **sh_allow_tcl_with_set_app_var** man page for details.
- The specified application variable is read only.
- The value specified is not a legal value for this application variable

EXAMPLES

The following example attempts to set a read-only application variable:

```
prompt> set_app_var synopsys_root /tmp
Error: can't set "synopsys_root": variable is read-only
       Use error_info for more info. (CMD-013)
```

In this example, the application variable name is entered incorrectly, which generates an error message:

```
prompt> set_app_var sh_enabel_page_mode 1
Error: "sh_enabel_page_mode" is not an application variable
        Use error_info for more info. (CMD-013)
```

This example shows the variable name entered correctly:

```
prompt> set_app_var sh_enable_page_mode 1
1
```

This example resets the variable to its default value:

```
prompt> set_app_var sh_enable_page_mode -default
0
```

SEE ALSO

```
get_app_var(2)
report_app_var(2)
write_app_var(2)
```

set_case_analysis

Specifies a constant logic value or a particular transition on a port or pin in the design, thereby restricting the scope of the analysis to the specified condition.

SYNTAX

```
status set_case_analysis
   value
   port_or_pin_list
   [-enable_trace_from]
   [-exclude_from_when]
```

Data Types

```
   value              string
   port_or_pin_list   list
```

ARGUMENTS

value

Specifies the constant logic value (0 or 1) that exists at the specified port or pin. The strings accepted for this option are **0**, **1**, **zero**, and **one**.

port_or_pin_list

The ports or pins to which the case analysis value applies.

-enable_trace_from

Directs NanoTime to ignore this case setting when path tracing from the specified input port.

-exclude_from_when

Excludes the specified input port from 'when' logic conditions that appear in extracted timing models.

DESCRIPTION

The **set_case_analysis** command specifies a constant logic value (either 0 or 1) on a port or pin in the design, thereby restricting the scope of the analysis to the specified condition.

For example, you can set a logic 0 on a test-mode input, and thereby eliminate consideration of the test mode circuitry. NanoTime propagates this constant value forward through the design as long as the value controls the value of the downstream logic. If you specify a constant logic 0 at the input of a NAND gate, the output of the NAND gate must be a constant logic 1, so NanoTime propagates this logic 1 through the output net. If this logic 1 feeds into an input of a NOR gate, NanoTime propagates a logic 0 at the output of the NOR gate, and so on.

NanoTime propagates the constant through combinational library cells (Synopsys Liberty models) using the function attributes specified in the library. NanoTime does not propagate a constant through a sequential cell function attribute. If you specify a logic constant on a pin, NanoTime propagates the logic constant forward from the pin, but not backward.

Case analysis can disable the library cell arcs and CCS receiver models based on the 'when' statement defined in the library. If the cell arc or the CCS receiver model has a 'when' condition, the condition is evaluated. The arc or the model will be disabled if the 'when' condition evaluates to false. Setting or removing logic values changes the design, making it necessary to run the **check_design** and **trace_paths** commands to get new analysis results.

NOTE: The Synopsys Design Constraints (SDC) format supports setting a rising or falling transition with the **set_case_analysis** command. However, this capability is not supported in NanoTime.

Command Phasing

The **set_case_analysis** command can only be executed between the "netlist-linked" and "topology-checked" states. If used after the "topology-checked" state, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The following command applies a constant logic 0 to the port IN1.

```
nt_shell> set_case_analysis 0 IN1
```

The following command applies a constant logic 1 to the pin Xreg51.X3.Mp0.g.

```
nt_shell> set_case_analysis one \
          [get_pins Xreg51.X3.Mp0.g]
```

SEE ALSO

remove_case_analysis(2)
report_case_analysis(2)
set_logic_constraint(2)

set_cle_options

Sets command line editor settings.

SYNTAX

```
status set_cle_options
   [-mode editor]
   [-beep on_off]
   [-defaults]
```

Data Types

```
   editor       string
   on_off       string
```

ARGUMENTS

-mode *editor*

Sets the command line editor mode. Valid argument values are **vi** or **emacs**.

-beep *on_off*

Enables or disables the command line editor beep sound. Valid argument values are **on** or **off**.

-defaults

Sets the command line editor to the default settings. This option overrides other options. The default edit mode is **emacs** and the default terminal beep setting is **off**.

DESCRIPTION

The **set_cle_options** command specifies command line editor settings.

If used without any options, it displays the current editor settings.

COMMAND PHASING

The **set_cle_options** command is not phase-restricted.

EXAMPLES

The following example sets the command line editor to vi mode.

```
nt_shell> set_cle_options -mode vi
Information: Command line editor mode is set to vi successfully. (CLE-01)
1
```

The following example enables the terminal beep.

```
nt_shell> set_cle_options -beep on
1
```

The following example sets the command line editor to the default settings.

```
nt_shell> set_cle_options -defaults
1
```

The following example displays the current command line editor settings.

```
nt_shell> set_cle_options

**************************************
Report : Command line editor settings
Version: Y-2006.06-Eng4
Date   : Wed May 10 13:26:37 2006
**************************************

Information: nt_shell is currently in emacs editing mode. (CLE-06)
Information: Terminal beep is disabled. (CLE-07)

1
```

SEE ALSO

sh_enable_line_editing(3)
sh_list_key_bindings(2)

set_clock_latency

Specifies the amount of delay from a clock source to sequential elements in the design.

SYNTAX

```
status set_clock_latency
   [-rise]
   [-fall]
   [-min]
   [-max]
   [-source]
   [-late]
   [-early]
   delay
   object_list
```

Data Types

```
   delay              float
   object_list        list
```

ARGUMENTS

-rise

Specifies the latency for rising-edge transitions.

-fall

Specifies the latency for falling-edge transitions.

-min

Specifies the latency for min operating conditions.

-max

Specifies the latency for max operating conditions.

-source

Specifies source latency rather than network latency. Source latency is the delay from the external

ideal waveform to the clock definition point in the design. Network latency is the delay from the clock definition point to sequential elements in the design.

-late

Specifies the maximum source latency value, which is used to calculate the latest clock arrival times. Can be used only with the **-source** option.

-early

Specifies the minimum source latency value, which is used to calculate the earliest clock arrival times. Can be used only with the **-source** option.

delay

Specifies the clock latency in time units. For source latency (with the **-source** option), it is the delay from the external ideal waveform to the clock definition point in the design. For network latency (without the **-source** option), it is the delay from the clock definition point to the listed objects.

object_list

A list of clocks, ports, or pins for which the latency value applies. For source latency, specify one or more clocks or clock source objects. For network latency, specify a list of clocks, ports, or pins. The network latency applies to all objects in the transitive fanout of the specified objects. For network latency set on a clock, the value applies to all elements clocked by that clock.

DESCRIPTION

Latency is the amount of time that a clock signal takes to get from its ideal-waveform origin point to a sequential element inside the design. Clock latency consists of two parts: source latency and network latency.

Source latency is the amount of time the clock signal takes to get from its ideal-waveform origin point to the clock definition point in the design, often called the clock source. The clock source is the port or pin specified in the **create_clock** command. Network latency is the amount of time the clock signal takes to get from the clock source object to sequential elements in the design.

Propagated and Ideal Clocking

For each sequential element in the design, NanoTime uses either propagated or ideal clocking. 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.

By default, NanoTime assumes that all clocks are propagated. To create clocks that are ideal, set the **timing_all_clocks_propagated** variable to **false** before you use the **create_clock** command. To use ideal clocking for specified clocks, ports, or pins, use the **set_clock_latency** command and set the desired latency values. To set ideal clocking on clocks, ports, or pins without applying a latency value (to use zero latency) use the **remove_propagated_clock** command. To restore propagated clocking behavior, use the **set_propagated_clock** command.

Source Latency and Network Latency

The **set_clock_latency** command specifies the amount of source latency for a clock or the amount of network latency for sequential elements in the design.

For source latency, use the **-source** option and specify the name of a clock or a clock source object in the design.

For network latency, do not use the **-source** option and specify a list of clocks, ports, or pins. If you specify a clock, it makes the clock ideal and sets the network latency of all sequential elements clocked by that clock. If you specify a port or pin, it sets the network latency of all objects in the transitive fanout of the port or pin, and makes those objects use ideal clocking.

You can use source latency to model off-chip clock latency when the clock generation circuit is not part of the current design. For a generated clock, you can use source latency to model the delay from the master clock to the generated clock.

You can specify source latency for both ideal and propagated clocks. However, specifying network latency (without the **-source** option) for a propagated clock changes that clock to ideal and issues a warning about the change.

For a generated clock, NanoTime automatically computes the clock source latency if the master clock has propagated latency and there is no user-specified value for the generated clock source latency. If the master clock is ideal and has source latency, and if there is no user-specified source latency value for the generated clock, then zero source latency is assumed.

Rise or Fall, Min or Max, Early or Late Options

By using the **-rise** or **-fall** option, you can specify the latency value for just rising edges or just falling edges of the clock signal. Without using these options, the specified latency value applies to both rising and falling edges. For source latency, the specified edge type applies to the clock signal at the clock source port or pin. For network latency, the specified edge type applies to the signal arriving at the sequential element, which can be different from the transition at the clock source due to logical inversions along the clock path.

If you are using different sets of operating conditions, you can optionally use the **-min** and **-max** options to set different latency values for min and max operating conditions.

To specify a worst-case range of source latency values, use the **-early** or **-late** option together with the **-source** option. NanoTime uses the early latency value to calculate the earliest possible clock arrival times, and the late latency value to calculate the latest possible clock arrival times. For example, for a setup check, it uses the late value for data launch and the early value for data capture.

To undo the effects of the **set_clock_latency** command, use the **remove_clock_latency** command.

To report network or source latency set on a clock, use the **report_clock** command with the **-skew** option. To report network or source latency set on a port or pin, use the **get_attribute** command. For example:

```
nt_shell> get_attribute [get_ports clk3] \
          clock_latency_fall_max
```

COMMAND PHASING

The **set_clock_latency** command can be used any time after the **link_design** command.

EXAMPLES

The following example specifies a rising-edge latency of 1.2 and a falling-edge latency of 0.9 for clock CLK1. Because the **-source** option is not used, each command sets the network latency of clock CLK1, which applies to all sequential elements clocked by CLK1. The edge type (rising or falling) refers to the signal transition arriving at the sequential element, including any inversions along the clock path.

```
nt_shell> set_clock_latency 1.2 \
          -rise [get_clocks CLK1]
nt_shell> set_clock_latency 0.9 \
          -fall [get_clocks CLK1]
```

The following example specifies an early source latency of 0.8 and a late source latency of 0.9 for clock CLK1. NanoTime uses the worst-case source latency value for each clock path of each timing check.

```
nt_shell> set_clock_latency 0.8 -source \
          -early [get_clocks CLK1]
nt_shell> set_clock_latency 0.9 -source \
          -late  [get_clocks CLK1]
```

SEE ALSO

```
remove_clock_latency(2)
remove_propagated_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_transition(2)
set_clock_uncertainty(2)
set_propagated_clock(2)
```

set_clock_period

Changes the period of an existing clock.

SYNTAX

```
status set_clock_period
   period_value
   object_list
```

Data Types

period_value	float
object_list	list

ARGUMENTS

period_value

The new clock period.

object_list

The list of clocks affected by the command.

DESCRIPTION

The **set_clock_period** command changes the period of an existing clock previously created by the **create_clock** command.

If you have already run the **trace_paths** command, using the **create_clock** command to change a clock causes NanoTime to go back to the clock propagation stage. This means that you must perform the **check_topology** command, timing constraint specification, and **check_design** command steps again before you can do any more path tracing.

If you only want to change the period of a clock, you can use the **set_clock_period** command instead,

which retains the clock propagation, topology recognition, and timing constraint information. The command takes you to the analysis stage just before the **trace_paths** command, similar to running the **reset_design -paths** command. You can immediately run the **trace_paths** command again using the new clock period.

The **set_clock_period** command cannot be used to change a generated clock directly, but it affects generated clocks that are based on the clock being changed. A virtual generated clock (which has no source pin) is updated immediately, whereas a generated clock with a source pin is updated during path tracing.

The **set_clock_waveform** command can be similarly used to change a clock waveform without returning to the clock propagation stage.

COMMAND PHASING

The **set_clock_period** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command changes the period of clock clk1 to 4.0 time units.

```
nt_shell> set_clock_period 4.0 [get_clocks clk1]
```

The following example demonstrates how to change both the period and waveform of a clock.

```
nt_shell> create_clock PHI1 -period 10.0 \
          -waveform { 5.0 9.5 }
 ...
nt_shell> set_clock_waveform {4.8 7.6} PHI1
nt_shell> set_clock_period 8.0 PHI1
```

In the foregoing example, the **set_clock_waveform** command must be executed before the **set_clock_period** command because the new clock period of 8.0 is inconsistent with the old waveform falling-edge time at 9.5.

SEE ALSO

```
create_clock(2)
report_clock(2)
reset_design(2)
set_clock_waveform(2)
trace_paths(2)
```

set_clock_transition

Specifies the transition time of clock signals reaching sequential elements in the design.

SYNTAX

```
status set_clock_transition
   [-rise]
   [-fall]
   [-min]
   [-max]
   transition
   clock_list
```

Data Types

```
   transition          float
   clock_list          list
```

ARGUMENTS

-rise

Specifies the transition time for rising clock edges.

-fall

Specifies the transition time for falling clock edges.

-min

Specifies the clock transition time for min operating conditions.

-max

Specifies the clock transition time for max operating conditions.

transition

Specifies the clock transition time, in time units of the technology.

clock_list

Specifies one or more clocks affected by the command. The transition time applies to all sequential elements clocked by the specified clock. The clock must have ideal latency. Otherwise, if the clock uses propagated latency, the transition time setting is ignored.

DESCRIPTION

The transition time of a signal (also known as slew) is the amount of time it takes for the signal to change from one logic state to the other. The transition time depends on the logic thresholds that define the logic 0 and logic 1 states. These thresholds can be set with the **rc_slew_*** variables.

The **set_clock_transition** command specifies the transition times of all clock signals reaching sequential elements clocked by a specified clock. This setting overrides transition times that are otherwise calculated from driver and load considerations.

The transition time setting only applies to an ideal clock (a clock with specified network latency settings). For a propagated clock, NanoTime calculates transition times from driver and load considerations and ignores the **set_clock_transition** setting. For information about ideal versus propagated clocks, see the man page for the **set_clock_latency** command.

If no transition time setting is specified for an ideal clock, NanoTime uses the transition times defined by the following variables:

```
Variable Name              Default
sim_transition_max_fall    0.05
sim_transition_max_rise    0.05
sim_transition_min_fall    0.05
sim_transition_min_rise    0.05
```

By using the **-rise** or **-fall** option, you can specify the transition time for just rising edges or just falling edges of the clock signal. Without these options, the specified transition time applies to both rising and falling edges. If you are using different sets of operating conditions, you can optionally use the **-min** and **-max** options to set different transition times for min and max operating conditions.

To undo the effects of the **set_clock_transition** command, use the **remove_clock_transition** command.

To list all clock transition values which have been set, use the **report_clock -skew** command.

COMMAND PHASING

The **set_clock_transition** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). The command cannot be executed before the **link_design** command. If you use the **set_clock_transition** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example makes clock CLK1 an ideal (not propagated) clock. Then it specifies a latency of 4.25, a rise transition time of 0.38, and a fall transition time of 0.25 for clock signals arriving at all sequential elements clocked by CLK1. The last command reports the latency and transition time settings.

```
nt_shell> remove_propagated_clock \
          [get_clocks CLK1]
nt_shell> set_clock_latency 4.25 \
          [get_clocks CLK1]
nt_shell> set_clock_transition 0.38 -rise \
          [get_clocks CLK1]
nt_shell> set_clock_transition 0.25 -fall \
          [get_clocks CLK1]
nt_shell> report_clock -skew
```

SEE ALSO

remove_clock_transition(2)
remove_propagated_clock(2)
report_clock(2)
set_clock_latency(2)
set_clock_uncertainty(2)
set_propagated_clock(2)
sim_transition_max_fall(3)
sim_transition_max_rise(3)
sim_transition_min_fall(3)
sim_transition_min_rise(3)

set_clock_uncertainty

Specifies the uncertainty (skew) of specified clock networks.

SYNTAX

```
status set_clock_uncertainty
   -from from_clock
      | -rise_from rise_from_clock
      | -fall_from fall_from_clock
   -to to_clock
      | -rise_to rise_to_clock
      | -fall_to fall_to_clock]
   [-rise]
   [-fall]
   [-setup]
   [-hold]
   [-same_cycle]
   [object_list]
   uncertainty
```

Data Types

```
   from_clock              list
   rise_from_clock         list
   fall_from_clock         list
   to_clock                list
   rise_to_clock           list
   fall_to_clock           list
   object_list             list
   uncertainty             float
```

ARGUMENTS

object_list

Specifies a list of clocks, ports, or pins for single-clock uncertainty. The uncertainty value is applied to capturing latches clocked by a clock in the *object list*, or to capturing latches whose clock pins are in the fanout of a port or pin specified in the *object_list*.

-from *from_clock*

Specifies the source clock for interclock uncertainty. You must specify either a pair of **-from**, **-**

rise_from, or **-fall_from** and **-to**, **-rise_to**, or **-fall_to** options, or an *object_list* argument, but not both.

-rise_from *rise_from_clock*

Same as the **-from** option, but indicates that the uncertainty value applies only to rising edges of the source clock. You can use only one of the **-from**, **-rise_from**, or **-fall_from** options.

-fall_from *fall_from_clock*

Same as the **-from** option, but indicates that the uncertainty value applies only to falling edges of the source clock.

-to *to_clock*

Specifies the destination clock for interclock uncertainty.

-rise_to *rise_to_clock*

Same as the **-to** option, but indicates that the uncertainty value applies only to rising edges of the destination clock. You can use only one of the **-to**, **-rise_to**, or **-fall_to** options.

-fall_to *fall_to_clock*

Same as the **-to** option, but indicates that the uncertainty value applies only to falling edges of the destination clock.

-rise

Applies the uncertainty value only to rising edges of the destination clock. By default, the uncertainty applies to both rising and falling edges. This option is valid only for interclock uncertainty. Unless you need this option for backward compatibility, use the **-rise_to** option instead.

-fall

Applies the uncertainty value only to falling edges of the destination clock. By default, the uncertainty applies to both rising and falling edges. This option is valid only for interclock uncertainty. Unless you need this option for backward compatibility, use the **-fall_to** option instead.

-setup

Applies the uncertainty value only to setup checks. By default, it applies to both setup and hold checks.

-hold

Applies the uncertainty value only to hold checks. By default, it applies to both setup and hold checks.

-same_cycle

Applies the uncertainty value only to same_cycle checks. By default, it applies to both next_cycle and same_cycle checks. Use this option to specify interclock uncertainty between same clock edges. This option does not have any effect when the **timing_propagate_interclock_uncertainty** variable is set to **true**.

uncertainty

A floating-point number that specifies the uncertainty value. Typically, clock uncertainty should be positive. Negative uncertainty values are supported for constraining designs with complex clock relationships. Setting the uncertainty value to a negative number could lead to optimistic timing analysis

and should be used with extreme care.

DESCRIPTION

The **set_clock_uncertainty** command specifies the uncertainty (skew characteristics) of specified clock networks. This command can specify either simple uncertainty (between different edges of the same clock) or interclock uncertainty (between two different clocks).

The command specifies the uncertainty as a floating-point value. This is the maximum amount of variation in arrival times for a single clock, or the maximum amount of skew between two different clocks, in the time units of the design. NanoTime uses the worst possible variation in clock arrival times to perform setup and hold checks. You should specify the worst expected uncertainty for the clock or between two clocks, or a larger number to provide additional timing margin.

For simple uncertainty, specify a list of clocks, ports, or pins. If you specify a clock, the uncertainty value applies to all sequential elements clocked by that clock. If you specify a port or pin, the uncertainty value applies to all sequential elements in the fanout of the port or pin.

For interclock uncertainty, use the **-from**, **-rise_from**, and **-fall_from** options to specify the source clock. Use the **-to**, **-rise_to**, and **-fall_to** options to specify the destination clock. The uncertainty value applies to all paths that start from the source clock domain and end in the destination clock domain, for the specified edge types.

For interclock uncertainty, be sure to specify all possible interactions of clock domains. For example, for paths from CLKA to CLKB and from CLKB to CLKA, you must specify the uncertainty for both directions, even if the uncertainty values are the same.

If there is no interclock uncertainty specified for a path, the value for simple uncertainty is used. A command that specifies interclock uncertainty has higher precedence than a conflicting command that specifies simple uncertainty.

If path-based slack adjustment is invoked by the **-pbsa** option of the **trace_paths** command, uncertainty values specified by the **set_clock_uncertainty** command are adjusted by the factors set by the **pbsa_KUsetup** and **pbsa_KUhold** variables. For more information about path-based slack adjustment, see the man page for the **report_pbsa_calculation** command.

To remove uncertainty values set by the **set_clock_uncertainty** command, use the **remove_clock_uncertainty** command.

To view the current clock uncertainty settings, use the **report_clock -skew** command.

NOTE: The **set_clock_uncertainty** command is a Synopsys Design Constraints (SDC) command. However, the **-rise_from**, **-rise_to**, **-fall_from** and **-fall_to** options might not be supported by other tools that use SDC commands. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_clock_uncertainty** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). The command

cannot be executed before the **link_design** command. If you use the **set_clock_uncertainty** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example specifies that all paths leading to elements clocked by CLK have setup uncertainty of 0.65 and hold uncertainty of 0.45.

```
nt_shell> set_clock_uncertainty -setup 0.65 \
          [get_clocks CLK]
nt_shell> set_clock_uncertainty -hold 0.45 \
          [get_clocks CLK]
```

The following example specifies interclock uncertainty within and between clock domains PHI1 and PHI2.

```
nt_shell> set_clock_uncertainty 0.4 \
          -from PHI1 -to PHI1
nt_shell> set_clock_uncertainty 0.4 \
          -from PHI2 -to PHI2
nt_shell> set_clock_uncertainty 1.1 \
          -from PHI1 -to PHI2
nt_shell> set_clock_uncertainty 1.1 \
          -from PHI2 -to PHI1
```

The following example specifies interclock uncertainty between clock domains PHI1 and PHI2 for specific edges.

```
nt_shell> set_clock_uncertainty 0.4 \
          -rise_from PHI1 -to PHI2
nt_shell> set_clock_uncertainty 0.4 \
          -fall_from PHI2 -rise_to PHI2
nt_shell> set_clock_uncertainty 1.1 \
          -from PHI1 -fall_to PHI2
```

The following example shows conflicting **set_clock_uncertainty** commands, one for simple uncertainty and one for interclock uncertainty. The interclock uncertainty value of 2.0 takes precedence for paths from CLKB to CLKA.

```
nt_shell> set_clock_uncertainty 5.0 \
          [get_clocks CLKA]
nt_shell> set_clock_uncertainty 2.0 \
          -from [get_clocks CLKB] \
          -to [get_clocks CLKA]
```

The following example specifies the uncertainty from CLKA to CLKB and from CLKB to CLKA. Notice that both must be specified, even though the value is the same for both.

```
nt_shell> set_clock_uncertainty 2.0 \
          -from [get_clocks CLKA] \
          -to [get_clocks CLKB]
nt_shell> set_clock_uncertainty 2.0 \
          -from [get_clocks CLKB] \
          -to [get_clocks CLKA]
```

The following example shows a situation in which simple uncertainty setting is used for interclock

uncertainty. The first command specifies a simple uncertainty of 5.0 for CLKA paths. The second command specifies an interclock uncertainty of 2.0 for paths from CLKB to CLKA. If there are paths between CLKA and other clocks (for example, CLKC, CLKD ...) for which interclock uncertainty has not been specified, the simple uncertainty for CLKA is used (in this case, 5.0).

```
nt_shell> set_clock_uncertainty 5.0 \
          [get_clocks CLKA]
nt_shell> set_clock_uncertainty 2.0 \
          -from [get_clocks CLKB] \
          -to [get_clocks CLKA]
```

SEE ALSO

remove_clock_uncertainty(2)
report_clock(2)
report_pbsa_calculation(2)
set_clock_latency(2)
set_clock_transition(2)
trace_paths(2)
pbsa_KUsetup(3)
pbsa_KUhold(3)
timing_propagate_interclock_uncertainty(3)

set_clock_waveform

Changes the waveform of an existing clock.

SYNTAX

```
status set_clock_waveform
   edge_list
   object_list
```

Data Types

```
   edge_list          list
   object_list        list
```

ARGUMENTS

edge_list

A list containing two values, the rising-edge time and falling-edge time, corresponding to the **-rise** and **-fall** option settings or **-waveform** option values of the original **create_clock** command.

object_list

The list of clocks affected by the command.

DESCRIPTION

The **set_clock_waveform** command changes the rising-edge time and the falling-edge time of an existing clock previously created by the **create_clock** command.

You must specify two values in a list and the name or names of the clocks to be changed. The first value is the new rising-edge time and the second value is the new falling-edge time. These values correspond to the **-rise** and **-fall** option settings or **-waveform** option values of the original **create_clock** command used to create the clock waveform.

If you have already run the **trace_paths** command, using the **create_clock** command to change a clock causes NanoTime to go back to the clock propagation stage. This means that you must perform the **check_topology** command, timing constraint specification, and **check_design** command steps again before you can do any more path tracing.

If you only want to change the clock waveform, you can use the **set_clock_waveform** command instead, which retains the clock propagation, topology recognition, and timing constraint information. The command takes you to the analysis stage just before the **trace_paths** command, similar to running the **reset_design -paths** command. You can immediately run the **trace_paths** command again using the new clock waveform.

The **set_clock_waveform** command cannot be used to change a generated clock directly, but it affects generated clocks that are based on the clock being changed. A virtual generated clock (which has no source pin) is updated immediately, whereas a generated clock with a source pin is updated during path tracing.

The **set_clock_period** command can be similarly used to change a clock period without returning to the clock propagation stage.

COMMAND PHASING

The **set_clock_period** command can be executed at any time after the **link_design** command.

EXAMPLES

The following command changes the waveform of clock clk2 to have its rising edge occur at time = 1.0 and its falling edge occur at time = 3.0.

```
nt_shell> set_clock_waveform {1.0 3.0} \
          [get_clocks clk2]
```

The following example demonstrates how to change both the period and waveform of a clock.

```
nt_shell> create_clock PHI1 -period 10.0 \
          -waveform { 5.0 9.5 }
 ...
nt_shell> set_clock_waveform {4.8 7.6} PHI1
nt_shell> set_clock_period 8.0 PHI1
```

In the foregoing example, the **set_clock_waveform** command must be executed first, before the **set_clock_period** command, because the new clock period of 8.0 is inconsistent with the old waveform falling-edge time at 9.5.

SEE ALSO

```
create_clock(2)
report_clock(2)
```

```
reset_design(2)
set_clock_period(2)
trace_paths(2)
```

set_conservative_max_delay

Specifies nets to be analyzed for a conservative maximum delay during user-guided multi-input switching analysis.

SYNTAX

```
status set_conservative_max_delay
   net_list
```

Data Types

```
   net_list     list
```

ARGUMENTS

net_list

The list of nets to be analyzed with simultaneous switching for maximum delay paths during user-guided multi-input switching analysis.

DESCRIPTION

The **set_conservative_max_delay** command affects user-guided multi-input switching analysis. This command requires a NanoTime Ultra license.

The command causes NanoTime to analyze maximum delay paths using the worst-case user-guided switching of inputs connected to the channel-connected regions of the specified list of nets. Side inputs configured as switching are assumed to switch simultaneously with the trigger input. Slew rates and voltage swing on the inputs are the same as the trigger.

The transistor direction is used to set up the devices for simultaneous switching.

To verify the result of this command, use the **write_spice** command and examine the nets included in the output.

To cancel the effects of the **set_conservative_max_delay** command, use the **remove_conservative_max_delay** command.

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**.

COMMAND PHASING

The **set_conservative_max_delay** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **set_conservative_max_delay** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example identifies all the nets whose names match "int*" as nets for simultaneous switching maximum delay.

```
nt_shell> set_conservative_max_delay int*
```

SEE ALSO

```
set_conservative_min_delay(2)
remove_conservative_max_delay(2)
remove_conservative_min_delay(2)
write_spice(2)
set_transistor_direction(2)
timing_enable_multi_input_switching(3)
```

set_conservative_min_delay

Specifies nets to be analyzed for a conservative minimum delay during user-guided multi-input switching analysis.

SYNTAX

```
status set_conservative_min_delay
   net_list
```

Data Types

```
   net_list     list
```

ARGUMENTS

net_list

The list of nets to be analyzed with simultaneous switching for minimum delay paths during user-guided multi-input switching analysis.

DESCRIPTION

The **set_conservative_min_delay** command affects user-guided multi-input switching analysis. This command requires a NanoTime Ultra license.

The command causes NanoTime to analyze minimum delay paths using the worst-case user-guided switching of inputs connected to the channel-connected regions of the specified list of nets. Side inputs configured as switching are assumed to switch simultaneously with the trigger input. Slew rates and voltage swing on the inputs are the same as the trigger.

The transistor direction is used to set up the devices for simultaneous switching.

To verify the result of this command, use the **write_spice** command and examine the nets included in the output.

To cancel the effects of the **set_conservative_min_delay** command, use the **remove_conservative_min_delay** command.

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**.

COMMAND PHASING

The **set_conservative_min_delay** command is typically executed in the "netlist-linked" state (in other words, between the **link_design** command and the **check_topology** command). If you use the **set_conservative_min_delay** command after the **check_topology** command, NanoTime transitions back to the "netlist-linked" state and discards all commands and operations performed after the **check_topology** command.

EXAMPLES

The following example identifies all the nets whose names match "int*" as nets for simultaneous switching minimum delay.

```
nt_shell> set_conservative_min_delay int*
```

SEE ALSO

```
set_conservative_max_delay(2)
remove_conservative_max_delay(2)
remove_conservative_min_delay(2)
write_spice(2)
set_transistor_direction(2)
timing_enable_multi_input_switching(3)
```

set_correlated_input

Specifies the correlated input with skew for a given pin.

SYNTAX

```
status set_correlated_input
   -pin pin_name
   -correlated_pin correlated_pin_name
   -skew skew_value
   [-slope slope_value]
   [-min]
   [-max]
```

Data Types

pin_name	string
correlated_pin_name	string
skew_value	float
slope_value	float

ARGUMENTS

-pin *pin_name*

Specifies the pin name

-correlated_pin *correlated_pin_name*

Specifies the correlated pin

-skew *skew_value*

Specifies the skew value in ns. Must be a positive floating point number.

-slope *slope_value*

Specifies the slope value in ns to use when driving the input of the correlated pin. Must be a positive floating point number.

-min

Specifies that the correlation is for the minimum delay path only.

-max

Specifies that the correlation is for the maximum delay path only.

DESCRIPTION

This command sets correlation between gate pins that have correlated behavior, usually with some common sources. In order to correctly analyze the skewed multi-input switching scenario, you can specify this correlation together with skews.

Use the **-min** or **-max** option to specify different worst-case minimum and maximum correlations.

Alternatively, you can use the **mark_correlated_region** command instead, to direct NanoTime to calculate the skew between correlated inputs.

COMMAND PHASING

The **set_correlated_input** command can be executed any time after the **link_design** command but before the **check_design** command.

SEE ALSO

mark_correlated_region(2)

set_current_command_mode

SYNTAX

```
string set_current_command_mode

   -mode command_mode | -command command

string command_mode
string command
```

ARGUMENTS

-mode *command_mode*

Specifies the name of the new command mode to be made current. **-mode** and **-command** are mutually exclusive; you must specify one, but not both.

-command *command*

Specifies the name of the command whose associated command mode is to be made current. **-mode** and **-command** are mutually exclusive; you must specify one, but not both.

DESCRIPTION

The **set_current_command_mode** sets the current command mode. If there is a current mode in effect, the current mode is first canceled, causing the associated clean up to be executed. The initialization for the new command mode is then executed as it is made current.

If the name of the given command mode is empty, the current command mode is cancelled and no new command mode is made current.

A current command mode stays in effect until it is displaced by a new command mode or until it is cleared by an empty command mode.

If the command completes successfully, the new current command mode name is returned. On failure,

the command returns the previous command mode name which will remain current and prints an error message unless error messages are suppressed.

EXAMPLES

The following example sets the current command mode to a mode called "mode_1"

```
shell> set_current_command_mode -mode mode_1
```

The following example clears the current command mode without setting a new mode.

```
shell> set_current_command_mode -mode ""
```

The following example sets the current command mode to the mode associated with the command "modal_command"

```
shell> set_current_command_mode -command modal_command
```

set_data_check

Creates nonsequential timing checks at specified locations in the design.

SYNTAX

```
status set_data_check
   [-label name]
    -from from_object
      | -rise_from from_object
      | -fall_from from_object
    -to to_object
      | -rise_to to_object
      | -fall_to to_object
   [-trigger trigger_object
      | -rise_trigger trigger_object
      | -fall_trigger trigger_object]
   [-setup | -hold]
   [-use_existing_timing_points]
   check_value
```

Data Types

```
name              string
from_object       list
to_object         list
trigger_object    list
check_value       list
```

ARGUMENTS

-label *name*

An optional label used to identify the timing check.

-from *from_object*

Specifies a pin or port in the current design as the reference pin of the nonsequential timing check. You must specify one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising transitions at the reference pin.

-fall_from ***from_object***

Similar to the **-from** option, but applies only to falling transitions at the reference pin.

-to ***to_object***

Specifies a pin or port in the current design as the checked pin of the nonsequential timing check. You must specify one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***to_object***

Similar to the **-to** option, but applies only to rising transitions at the checked pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling transitions at the checked pin.

-trigger ***trigger_object***

The trigger port or pin at the input of the simulation unit related to the timing check, where path tracing starts. This option can only be used if the **-use_existing_timing_points** option is also set.

-rise_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to rising signals at the trigger object.

-fall_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to falling signals at the trigger object.

-setup

Indicates that the check value is for setup checking only.

-hold

Indicates that the check value is for hold checking only. If neither of the **-setup** or **-hold** options is specified, the value applies to both setup and hold checks.

-use_existing_timing_points

When this option is used the timing check creation must refer only to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used then the **check_design** command does not need to be run again after the **set_data_check** command has been issued.

check_value

Specifies the setup or hold time for the check.

DESCRIPTION

The **set_data_check** command creates a nonsequential timing check, sometimes called a data check, to

be performed between a "from" object (also known as the reference port or pin) and a "to" object (also known as the checked port or pin). You can create a data check between two clock objects or between two data objects. This feature can be useful for checking constraints on handshaking interface logic, asynchronous interfaces, signals with unusual clock waveforms, or skew between bus lines.

The pulse relation between the clocks of the related pin and the constrained pin is considered to be zero cycles (same-cycle checking).

Data checks are performed only after path tracing is complete.

A timing check specified by this command is performed in addition to any other checks already being performed between the "from" and "to" points. It does not replace any existing checks. After running the **check_design** command, you can modify a timing check with the **set_timing_check_attributes** command. You can remove a data check with the **remove_data_check** command.

NOTE: The **set_data_check** command is a Synopsys Design Constraints (SDC) command. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_data_check** command is typically executed in the "netlist-linked" or "topology-checked" state (in other words, between the **link_design** and **check_design** commands). You cannot use the command before the **link_design** command. If you use the **set_data_check** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option, you can use the **set_data_check** command after the **check_design** command has been executed.

EXAMPLES

The following example creates a data check from pin ac1/B to pin ac1/A with respect to the rising edge of the signal at ac1/B, using a setup and hold time of 0.4.

```
nt_shell> set_data_check -rise_from ac1/B \
          -to ac1/A 0.4
```

SEE ALSO

create_timing_check(2)
remove_data_check(2)
report_paths(2)
set_timing_check_attributes(2)
trace_paths(2)

set_data_trace_from_clock

Enables data tracing from the startpoints of specified clocks.

SYNTAX

```
status set_data_trace_from_clock
   clock_list
```

Data Types

```
   clock_list            list
```

ARGUMENTS

clock_list

Specifies a collection of clocks from whose startpoints NanoTime should perform data tracing.

DESCRIPTION

During path tracing, NanoTime performs data tracing from all user-defined inputs and clock tracing from all clock startpoints. Clock tracing finds each path that starts at the boundary of the design and ends at the clock input of a sequential element such as a latch, precharge structure, or RAM cell.

In addition, by default, NanoTime also performs data tracing from all clock startpoints. Data tracing finds each path that starts at a clock port, goes from the clock input to the Q output of a latch, and continues to another latch or to the output of the design.

To prevent NanoTime from performing data tracing for a given clock, use the **remove_data_trace_from_clock** command to specify the clock. If you later want to cancel the effects of this command and enable data tracing of paths for a given clock, use the **set_data_trace_from_clock** command. The default behavior is to perform data tracing for all clocks.

COMMAND PHASING

The **set_data_trace_from_clock** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_data_trace_from_clock** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command prevents data tracing from clock startpoints of clock ck1:

```
nt_shell> remove_data_trace_from_clock [get_clocks ck1]
1
```

The following command enables data tracing from clock startpoints of clock ck1, thus restoring the default behavior:

```
nt_shell> set_data_trace_from_clock [get_clocks ck1]
1
```

SEE ALSO

remove_data_trace_from_clock(2)
trace_paths(2)

set_dcs_input

Specifies a constant logic value on a net, port, or pin that is acting as a side input for dynamic clock simulation (DCS). The side input is still switchable for non-DCS path tracing.

SYNTAX

```
status set_dcs_input
   [-logic logic_value]
   [-synchronized_data]
   object_list
```

Data Types

logic_value	integer
object_list	list

ARGUMENTS

-logic *logic_value*

Specifies a constant logic value (0 or 1) to use on the specified net, port, or pin.

-synchronized_data

Specifies signals in self-timed circuits. These signals are inputs of the dynamic clock simulation region and drive channel-connected blocks whose outputs are clock nets.

object_list

The nets, ports, or pins to which the logic value applies.

DESCRIPTION

The **set_dcs_input** command specifies a constant logic value (0 or 1) on a net, port, or pin to restrict the side input logic values for dynamic clock simulation.

NanoTime uses dynamic clock simulation to get accurate clock arrivals. When setting up the simulation region, if the tool cannot determine the logic state for a side input, it creates a waveform for the side input so that both low and high states on the side input are covered by the simulation. A DCS-001 warning message is issued for each side input waveform. Since the waveforms must cover all possible combinations of side input logic states, this might have a large performance impact. In addition, incorrect results might arise from simulating logically impossible side input combinations. For better performance, use the **set_dcs_input** command with the **-logic** option to set side input logic.

The **set_case_analysis** command also affects the side input state. However, using case analysis disables path tracing and timing checks through the side input. In contrast, the **set_dcs_input** command only affects dynamic clock simulation analysis. It does not affect case analysis and the side input remains switchable for path tracing outside of dynamic simulation. The **set_dcs_input** command does not cause logic implications on other nets.

If you apply both the **set_dcs_input** and **set_case_analysis** commands to the same net, port, or pin, the **set_dcs_input** value takes precedence during dynamic simulation.

If you apply multiple **set_dcs_input** commands to the same net, port, or pin, the first one succeeds and the later ones fail.

The **-synchronized_data** option identifies a signal used in self-timed circuits.

COMMAND PHASING

The **set_dcs_input** command must be executed after the **check_topology** command but before the **check_design** command. If you use the **set_dcs_input** command after the **check_design** command, NanoTime discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following **set_dcs_input** command applies a constant logic 0 to net xccarray/nsel0 to resolve the DCS-001 message.

```
nt-shell>check_design
Starting dynamic clock simulation...
Warning: Waveform generated for undefined side input xccarray/nsel0,
which may have performance impact or result in incorrect output
waveforms. (DCS-001)
...
Finished dynamic clock simulation
Checking timing graph for consistency...
1

nt-shell>set_dcs_input -logic 0 xccarray/nsel0
1

nt-shell>check_design
Starting dynamic clock simulation...
Finished dynamic clock simulation
Checking timing graph for consistency...
1
```

The following example shows that for dynamic simulation, the logic specified by the **set_dcs_input** command takes precedence over the logic set with the **set_case_analysis** command or clock-gater. A warning message is reported after the **check_design** command when conflicts arise.

```
nt_shell> set_case_analysis 1 IN1
nt-shell> set_dcs_input -logic 0 IN1
1

nt-shell> check_design
Starting dynamic clock simulation...
Warning: DCS side input logic setting '0' conflicts with case
analysis settings on port IN1. (DCS-012)
Finished dynamic clock simulation
Checking timing graph for consistency...
1
```

The following example shows that if multiple **set_dcs_input** commands are applied to the same net, port, or pin with conflicting logic values, NanoTime uses the first setting and issues an error message for the later commands.

```
nt-shell> set_dcs_input -logic 0 xccarray/nsel0
1
nt-shell> set_dcs_input -logic 1 xccarray/nsel0
Error: DCS side input logic setting conflicts with previous DCS side
input logic setting on net xccarray/nsel0. (DCS-013)
0
```

The following example specifies the net sae_off as synchronized data. If sae_off is not an input of the dynamic simulation region, a warning message is issued during the check-design operation.

```
nt-shell> set_dcs_input -synchronized_data sae_off
1
```

SEE ALSO

```
remove_dcs_input(2)
set_case_analysis(2)
remove_case_analysis(2)
report_case_analysis(2)
set_logic_constraint(2)
```

set_delay_coefficients

Modifies the delay times of delay arcs associated with the specified trigger or target objects.

SYNTAX

```
status set_delay_coefficients
   [-min]
   [-max]
   [-min_clock]
   [-max_clock]
   [-rise]
   [-fall]
   [-delay delay_factor]
   [-transition transition_factor]
   [-offset offset_value]
   [-target target_objects]
   [trigger_objects]
```

Data Types

```
   delay_factor            float
   transition_factor       float
   offset_value            float
   target_objects          list
   trigger_objects         list
```

ARGUMENTS

-min

Modifies delay times for short data path analysis.

-max

Modifies delay times for long data path analysis.

-min_clock

Modifies delay times for short clock path analysis.

-max_clock

Modifies delay times for long clock path analysis.

-rise

Modifies delay times for rising input slope.

-fall

Modifies delay times for falling input slope.

-delay *delay_factor*

The delay factor, a positive number. NanoTime multiplies the original calculated delay by this factor to obtain the new delay. The default delay factor is 1.0.

-transition *transition_factor*

The transition factor. NanoTime multiplies the original calculated transition time at the input trigger pin by this factor to obtain a transition offset factor, which is added to the delay from the trigger to the output. The default transition factor is zero.

-offset *offset_value*

The offset value, a positive or negative number. NanoTime adds this number to the calculated delay. The default offset is zero.

-target *target_objects*

The list of nets, transistor gate pins, or output ports affected by the command. This list defines a set of delay calculation target pins, nets, or ports. If target objects are specified without any trigger objects, all delays to the target are modified. If both trigger objects and target objects are specified in the same command, the delay for the trigger-target pair is modified. If separate trigger-only and target-only commands are specified, the target-only coefficient takes precedence over a trigger-only coefficient. A command for a specific trigger-target pair takes precedence over any other command.

trigger_objects

The list of trigger nets or transistor gate pins affected by the command. This list defines a set of delay calculation trigger pins or nets. The delays from these trigger pins or nets to a driven output net are modified.

DESCRIPTION

The **set_delay_coefficients** command modifies the calculated delay values at specified locations in the design. At each specified trigger net or transistor gate, NanoTime modifies the calculated delay as follows:

```
new_delay = (calculated_delay * delay_factor) +
            (input_transition_time * transition_factor) + offset_value
```

This feature can be used to adjust delays for effects not already modeled by the operating conditions.

In the command, you must specify the types of timing arcs affected by the command (min data, max data, min clock, or max clock) and a list of objects that define the delay arcs to be modified. You can specify the affected delay arcs with a list of trigger objects as an argument to the command, a list of target objects as an argument to the **-target** option, or both.

You must also specify a delay factor, a transition factor, or offset value, or some combination of those parameters. If the result of modification is a negative number, NanoTime uses a delay of zero.

To remove delay coefficients that have been set, use the **remove_delay_coefficients** command.

COMMAND PHASING

The **set_delay_coefficients** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_delay_coefficients** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command modifies the delay for arcs starting at net Sn04 for max-path delay checking.

```
nt_shell> set_delay_coefficients -max \
          -delay 1.2 -transition 0.2 \
          -offset 0.5 [get_nets Sn04]
```

NanoTime first calculates the delay at net Sn04 normally, then multiplies the result by 1.2, then multiplies the input transition time by 0.2 and adds that result to the delay, and then adds the offset of 0.5 to the delay. For example, if the calculated delay is 4.0 and the input transition time is 1.0, the new delay is calculated as follows:

```
new_delay = (4.0*1.2) + (1.0*0.2) + 0.5
          = 4.8 + 0.2 + 0.5
          = 5.5
```

SEE ALSO

```
remove_delay_coefficients(2)
report_delay_coefficients(2)
remove_transition_coefficients(2)
set_transition_coefficients(2)
```

set_differential

Defines two objects of the same type (nets, pins, or ports) to be a differential pair.

SYNTAX

```
status set_differential
   differential_pair
   [-skew_max skew_value1]
   [-skew_min skew_value2]
```

Data Types

```
   differential_pair        list
   skew_value1              float
   skew_value2              float
```

ARGUMENTS

differential_pair

A list or collection containing exactly two objects of the same type: nets, ports, or pins.

-skew_max *skew_value1*

Specifies a constant skew (in the unit of ns) between two nets in a differential pair, to be used for maximum delay path tracing. The default is 0.

-skew_min *skew_value2*

Specifies skew (in the unit of ns) between two nets in a differential pair, to be used for minimum delay path tracing for path-based slack adjustment. The default is 0.

DESCRIPTION

The **set_differential** command defines two objects of the same type (nets, pins, or ports) to be a differential pair. NanoTime guarantees a differential pair to have inverted logic. The first item in the list is

the reference object and the second item is the complement object.

If this command is used to set differential pins or ports, the reference object must be either (1) a clock port or leaf pin defined by a **create_clock** or **create_generated_clock** command or (2) a leaf pin of a master clock source defined by a **create_generated_clock** command. The complement pin or port must not be a clock object. The logical nets associated with the two pins or ports are automatically set to be differential pairs.

If this command is used to define differential nets, you can optionally define skew between the two nets by using the **-skew_max** option. The specified skew must be greater than or equal to zero. The complement net always switches later than the trigger net, which means that the minimum skew for delay calculations is zero and the **-skew_min** option is ignored.

Manually setting the skew is unlikely to be accurate. NanoTime can automatically analyze skew by running path tracing iterations to dynamically refine skew values throughout the entire design. To enable this feature, set the **timing_differential_iteration_count** variable to a value greater than 1 and do not use the **-skew_max** option.

When path-based slack adjustment is enabled, both the **-skew_max** and **-skew_min** options apply to the delay calculations.

COMMAND PHASING

The **set_differential** command is typically used after the **link_design** command but before the **check_topology** command.

However, the command can be used later if you want to investigate the effects of changing the skew between differential nets. To do this, first run path tracing, then use the **reset_design -paths** command to reset the analysis database. You can change the constant net skew with the **set_differential -skew_max** command and run path tracing again.

EXAMPLES

The following command defines the two nets in3p and in3n to be differential nets, with in3p as the reference object, and sets the skew between them to be 3 for maximum delay analysis.

```
nt_shell> set_differential {in3p in3n} -skew_max 3
```

SEE ALSO

```
remove_differential(2)
timing_differential_iteration_count(3)
mark_differential_synchronizer(2)
erase_differential_synchronizer(2)
```

set_disable_logic_check

Specifies the nets in the design that cannot support static logic states during simulation.

SYNTAX

```
status set_disable_logic_check
   nets
```

Data Types

```
   nets          list
```

ARGUMENTS

nets

The list of nets that cannot support static logic states for simulation.

DESCRIPTION

NanoTime sets logic states on nets for simulation during path tracing. However, for some types of circuits such as pulse clock generators, it is not possible to sensitize certain nets to a specific static logic state. To allow analysis of the circuit, you must disable the logic checks on those nets with the **set_disable_logic_check** command.

Use the **remove_disable_logic_check** command to undo the effect of the **set_disable_logic_check** command.

COMMAND PHASING

The **set_disable_logic_check** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_disable_logic_check** command after the **check_design** command, NanoTime transitions back to

the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The current design has a pulse generator circuit with a net pgen1 that cannot be sensitized to a static logic 1, causing an error. The following command disables logic checking of that net, allowing analysis of the design to proceed.

```
nt_shell> set_disable_logic_check pgen1
```

SEE ALSO

remove_disable_logic_check(2)

set_drive

Specifies the resistance of the external driver on specified input ports and bidirectional ports in the current design.

SYNTAX

```
status set_drive
   [-rise]
   [-fall]
   [-min]
   [-max]
   resistance_value
   port_list
```

Data Types

resistance_value	float
port_list	list

ARGUMENTS

-rise

Applies only to rising transitions.

-fall

Applies only to falling transitions.

-min

Applies only to minimum delay calculation.

-max

Applies only to maximum delay calculation.

resistance_value

Specifies the resistance of the driver that is driving the specified ports, in the resistance units of the technology. This value must be greater than or equal to 0.0.

port_list

A list of input and bidirectional ports in the current design on which the driver resistance value is set.

DESCRIPTION

This command sets the resistance of the driver that is driving one or more input ports or bidirectional ports in the current design. In the command, you must specify the resistance value (in the resistance units of the technology) and the input ports or bidirectional ports in the design to which that resistance value applies.

NanoTime models the external driver as the voltage supply connected to a resistor for a rising transition, or ground connected to a resistor for a falling transition. The tool uses the resistance value and the capacitive load of the port to calculate the wire delay of the port. A higher drive resistance means less drive capability and a longer delay at the input port. Conversely, a drive resistance of zero results in no external delay.

If a transition time has been set on the port with the **set_input_transition** command, NanoTime models the external driver as a voltage ramp driving the resistor and connected to the port.

Use the **-min** option to specify the resistance value for minimum delay calculation only, or the **-max** option to specify the resistance value for maximum delay calculation only. Two **set_drive** commands are necessary to specify the resistance values for both minimum and maximum delay calculation.

You can use the **-rise** or **-fall** option to apply the resistance value to rising or falling edges only. Otherwise, the command applies to both rising and falling edges.

To view the drive resistance that has been set on a port, use the **report_port -drive** command.

To remove the drive resistance previously set on a port, use the **remove_drive_resistance** command.

COMMAND PHASING

The **set_drive** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_drive** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example sets the drive resistance to 0.25 on all input ports in the current design. The **report_port -drive** command shows the ports and their drive resistance values.

```
nt_shell> set_drive 0.25 [all_inputs]
1
```

```
nt_shell> report_port -drive
 ...
Resistance

Input Port   Min Rise Min Fall Max Rise Max Fall
------------ -------- -------- -------- --------
BUS[0]          0.250    0.250    0.250    0.250
BUS[1]          0.250    0.250    0.250    0.250
BUS[2]          0.250    0.250    0.250    0.250
BUS[3]          0.250    0.250    0.250    0.250
overflow        0.250    0.250    0.250    0.250
rshft           0.250    0.250    0.250    0.250
selA            0.250    0.250    0.250    0.250
selB            0.250    0.250    0.250    0.250
 ...
```

SEE ALSO

```
report_port(2)
set_load(2)
```

set_driving_cell

Sets driving cell information on input or inout ports.

SYNTAX

```
string set_driving_cell
   [-lib_cell lib_cell_name]
   [-pin pin_name]
   [-from_pin from_pin_name]
   [-input_transition_rise rtran]
   [-input_transition_fall ftran]
   port_list
```

Data Types

```
   lib_cell_name     string
   pin_name          string
   from_pin_name     string
   rtran             float
   ftran             float
   port_list         list
```

ARGUMENTS

-lib_cell *lib_cell_name*

Specifies the name of the library and name of library cell used to drive the ports as library.libcell. You must use the **-pin** option if the cell has more than one output pin. If different cells are needed for the rising and the falling cases, use separate commands with the **-rise** or **-fall** option. Use the **-from_pin** option to choose between multiple input pins with arcs to this output pin. This option is required.

-pin *pin_name*

Specifies the output pin of the lib_cell to be used to drive the port. This is the driving pin name.

-from_pin *from_pin_name*

Specifies an input pin on the specified cell so the command uses the drive of the timing arc from this pin to the specified pin.

-input_transition_rise *rtran*

Specifies the input rising transition time associated with the **-from_pin** option. If you do not include this option, the default value is 0. Use the **-input_transition_rise** and **-input_transition_fall** options to capture the accurate transition time associated with the *from_pin_name* value. This can obtain more accurate information on the transition time and delay time at the output pin.

-input_transition_fall ***ftran***

Specifies the input falling transition time associated with the *from_pin_name* value. If you do not include this option, the default is 0.

port_list

Provides a list of input ports. The list contains input or inout port names in the current design on which the driving cell information is set.

DESCRIPTION

The **set_driving_cell** command sets attributes on the specified input or inout ports in the current design to associate an external driving cell with the ports.

The drive capability of the port is the same as if the specified driving cell were connected in the same context to allow accurate modeling of port drive capability for nonlinear delay models.

There are two other methods of describing port drive capability: the **set_drive** command or the **set_input_transition** command. The most recent drive command has precedence. If possible, use the **set_driving_cell** command instead of the **set_drive** command, because the **set_driving_cell** command allows accurate calculation of port delay and transition time for library cells with nonlinear dependence on capacitance.

EXAMPLES

```
nt_shell> set_driving_cell \
-from_pin A -pin Z -lib_cell mylib.NOR -input_transition_rise 0.05 \
   -input_transition_rise 0.03 [get_ports in1]
```

SEE ALSO

set_drive(2)
set_input_transition(2)

set_enable_input_spf_skew

Enables back-annotated RC parasitics skew for a net and specifies the threshold for using it in delay calculations.

SYNTAX

```
status set_enable_input_spf_skew
   -threshold value
   nets
```

Data Types

```
   value        float
   nets         list
```

ARGUMENTS

-threshold

Specifies the threshold used to determine whether the skew is large enough to have delay impact. The value is in default time units.

nets

Specifies the list of nets affected by the command.

DESCRIPTION

By default, NanoTime analysis ignores skew caused by parasitic RC delay between connected trigger devices (boundary nodes of the same back-annotated RC network). However, the skew can be large enough to affect the accuracy of timing analysis. The **set_enable_input_spf_skew** command allows you to set a threshold for evaluating the magnitude of this RC delay for the specified nets.

If the value exceeds the threshold, timing analysis takes the input skew into consideration. If the skew

values between the inputs are less than the specified threshold, NanoTime saves the stage delay from the first input and reuses it for the other inputs, which is the same as the default behavior (when the command is not used).

It is important to select a small enough value for the threshold to avoid nonmonotonic behavior in a generated timing model by considering model input transition indexes and the separation between them.

COMMAND PHASING

Use the **set_enable_input_spf_skew** command after the **check_topology** command, but before the **trace_paths** command.

EXAMPLES

The following command enables the calculation of skew effect on net Sn32 with a threshold of 5 ps.

```
nt_shell> set_enable_input_spf_skew \
          -threshold 0.005 [get_nets Sn32]
```

SEE ALSO

report_net(2)
trace_paths(2)

set_exclude_reference_path

Specify a launch exception that when matched will suppress any capture paths matched by the -exclude_reference_through option from being used to check against.

SYNTAX

```
status set_exclude_reference_path
   -exclude_reference_through exclude_reference_through_list
      | -exclude_reference_rise_through exclude_reference_rise_through_list
      | -exclude_reference_fall_through exclude_reference_fall_through_list
   [-max | -min]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-use_existing_timing_points]
```

Data Types

```
   exclude_reference_through_list      list
   exclude_reference_rise_through_list list
   exclude_reference_fall_through_list list
   from_list                           list
   rise_from_list                      list
   fall_from_list                      list
   through_list                        list
   rise_through_list                   list
   fall_through_list                   list
   to_list                             list
   rise_to_list                        list
   fall_to_list                        list
```

ARGUMENTS

-exclude_reference_through *exclude_reference_through_list*

Specifies the nets or pins through which reference paths are excluded from being checked against any launch paths matching the **set_exclude_reference_path** exception. Clock and port objects are not

supported. Only one level of -exclude_reference_through must be specified per exception. Either the **-exclude_reference_through**, **-exclude_reference_rise_through**, or **-exclude_reference_fall_through** option must be specified.

-exclude_reference_rise_through ***exclude_reference_rise_through_list***

Same as the **-exclude_reference_through** option, except that only reference paths with rising-edge signals through the exclude_reference point are considered. Either the **-exclude_reference_through**, **-exclude_reference_rise_through**, or **-exclude_reference_fall_through** option must be specified.

-exclude_reference_fall_through ***exclude_reference_fall_through_list***

Same as the **-exclude_reference_through** option, except that only reference paths with falling-edge signals through the exclude_reference point are considered. Either the **-exclude_reference_through**, **-exclude_reference_rise_through**, or **-exclude_reference_fall_through** option must be specified.

-max

Only excludes reference paths in max launch path search.

-min

Only excludes reference paths in min launch path search.

-from ***from_list***

Specifies a list of path startpoint objects. A valid timing startpoint is a clock, a primary input or bidirectional port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has an input delay specified. If you specify a clock, the command affects all paths that start at sequential elements and primary inputs clocked by that clock. If you specify a cell, the command affects all paths that start at input pins and bidirectional pins of the cell. You can use at most one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only paths with rising-edge signals from the specified objects are affected. If you specify a clock, the command affects all paths that start at sequential elements and primary inputs clocked by the rising edge of that clock at the source (which could result in a rising or falling edge at the path startpoint, depending on logical inversions along the clock path.)

-fall_from ***fall_from_list***

Same as the **-rise_from** option, except that only paths with falling-edge signals from the specified objects are affected.

-through ***through_list***

Specifies a list of pins, ports, cells, or nets through which the paths must pass to be affected by the command. You can specify the **-through** option more than one time in the same command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net.

-rise_through ***rise_through_list***

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through ***fall_through_list***

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to ***to_list***

Specifies a list of path endpoint objects. A valid timing endpoint is a clock, a primary output or bidirectional port, a sequential cell, a data pin of a sequential cell, or a pin that has an output delay specified. If you specify a clock, the command affects all paths that end at sequential elements and primary outputs clocked by that clock. If you specify a cell, the command affects all paths that end at input pins and bidirectional pins of the cell. You can use at most one of the **-to**, **-rise_to**, and **-fall_to** options. Path endpoint objects only apply to the launch paths.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only paths with rising-edge signals to the specified objects are affected. If you specify a clock, the command affects all paths that end at sequential elements and primary outputs clocked by the rising edge of that clock at the source (which could result in a rising or falling edge at the path endpoint, depending on logical inversions along the clock path.)

-fall_to ***fall_to_list***

Same as the **-rise_to** option, except that only paths with falling-edge signals arriving at the specified objects are affected.

-use_existing_timing_points

When this option is used the exception definition must only refer to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **set_exclude_reference_path** command has been issued.

DESCRIPTION

The **set_exclude_reference_path** command can be used to exclude reference (capture) paths passing through pins or nets from launch (checked) paths that match the exception. This exception applies to both in-trace and user timing checks.

NanoTime will evaluate timing checks within a design. When this process is done, by default it will take the worst-case path to the checked point (launch path) and compare it to the worst-case path to the reference point (capture path). The launch and capture paths are traced independently and do not account for any possible restrictions that might have excluded these two paths from coinciding at the same time. The **set_exclude_reference_path** exception command allows the tool to account for launch and capture paths that cannot coexist.

This command defines a path-based timing exception, along with the **set_multicycle_path**, **set_no_same_phase_path**, **set_same_phase_path**, **set_false_path**, and **set_no_timing_check_path** commands. The **set_exclude_reference_path** exception works in conjunction with all of these. A false_path exception that matches anywhere along the exclude_reference capture or launch path takes precedence and stops the path tracer. A no_timing_check_path exception

that matches at the launch path checked point continues to ensure that no timing check evaluations occur whether or not the launch path also fully matched an exclude_reference exception.

The **-exclude_reference_through** option only supports nets and pins, and will not accept clocks or ports. Only one level of **-exclude_reference_through** can be specified per exception. Multiple exclude_reference exception commands along the same launch or capture path are not supported. And DCS (dynamic clock simulation) is not supported - if DCS is used, stop the clock network before the exclude_reference point. If any of these limitations are violated, the tool will issue a warning.

If all reference arrivals are excluded for a launch path that matches a **set_exclude_reference_path** exception, a warning will be issued and any timing check at the checked point classified as untested.

To undo the effects of a **set_exclude_reference_path** command, use the **remove_exclude_reference_path** command.

From, Through, and To Points

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any one of the listed objects. The **-consecutive** option specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

To restrict the exclude_reference exception to paths with only rising edges or only falling edges at the path endpoint, use the **-rise** or **-fall** option. These options can be used even when there is no "to" type option specified.

NOTE: The **set_exclude_reference_path** command has no corresponding Synopsys Design Constraints (SDC) command.

COMMAND PHASING

The **set_exclude_reference_path** command is typically used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the topology-checked state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option with the **set_exclude_reference_path** command, you can execute it after the **check_design** command and before the **trace_paths** command or **extract_model** command.

EXAMPLES

The following example excludes capture paths passing through net clkb with min launch paths passing through clka and net end. Any min (hold) timing checks at launch path endpoints matching this exception will use worst-case (max) reference arrivals excluding any capture paths that passed through net clkb.

```
nt_shell> set_exclude_reference_path -min \
          -through clka -exclude_reference_through clkb -through end
```

SEE ALSO

```
remove_exclude_reference_path(2)
report_exceptions(2)
set_false_path(2)
set_multicycle_path(2)
set_no_same_phase_path(2)
set_no_timing_check_path(2)
set_same_phase_path(2)
```

set_extended_sidebranch_level

Specifies the effort level to be used in the side branch exploration algorithm for specific nets.

SYNTAX

```
status set_extended_sidebranch_level
   objects
   level
```

Data Types

```
   objects       list
   level         integer
```

ARGUMENTS

objects

Specifies the nets to which the setting applies.

level

Specifies the effort level to be used in the side branch exploration algorithm. The default is 0. Valid values are 0, 1, 2, and 3.

DESCRIPTION

The **set_extended_sidebranch_level** command specifies the effort level to be used in the sidebranch exploration algorithm for the stages to which the specified nets belong. A net-specific effort level overrides the global effort level set with the **timing_extended_sidebranch_analysis_level** variable.

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. While this approach significantly improves the performance and capacity of the tool, it might not provide the desired level of accuracy in certain situations.

Using a high global effort level might lead to a significant increase in runtime and memory. Instead, set a high effort level only on specific nets.

The effort levels are as follows:

0: The default; allows analysis of up to 16 side inputs

1: Allows analysis of up to 32 side inputs

2: Allows analysis of up to 128 side inputs

3: Unlimited sidebranch exploration

COMMAND PHASING

The **set_extended_sidebranch_level** command must be executed after the **link_design** command but before the **check_design** command. If you use the **set_extended_sidebranch_level** command after the **check_design** command, NanoTime discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command sets the side branch level to 1 for the net soutn.

```
nt_shell> set_extended_sidebranch_level \
          [get_nets soutn] \
           1
```

SEE ALSO

timing_extended_sidebranch_analysis_level(3)

set_false_path

Identifies false paths in a design, which are not to be considered during timing analysis.

SYNTAX

```
status set_false_path
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-consecutive]
   [-use_existing_timing_points]
```

Data Types

```
   from_list            list
   rise_from_list       list
   fall_from_list       list
   through_list         list
   rise_through_list    list
   fall_through_list    list
   to_list              list
   rise_to_list         list
   fall_to_list         list
```

ARGUMENTS

-setup

Specifies that path tracing should not be performed for maximum delay analysis. If you do not specify either the **-setup** or **-hold** option, all path tracing is disabled.

-hold

Specifies that path tracing should not be performed for minimum delay analysis.

-rise

Specifies that paths with rising delays at the path endpoint are to be marked as false paths. If you do not specify either of the **-rise** or **-fall** options, both rise and fall timing are marked false.

-fall

Specifies that paths with falling delays at the path endpoint are to be marked as false paths.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all sequential elements and primary inputs related to that clock are used as path startpoints. If a cell is specified, all input pins and bidirectional pins of that cell are affected. You can use at most one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only rising-edge signals from the specified objects are affected. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from ***fall_from_list***

Same as the **-rise_from** option, except that only falling-edge signals from the specified objects are affected.

-through ***through_list***

Specifies a list of pins, ports, cells, or nets through which the paths must pass. You can specify **-through** more than one time in one command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net. If you do not specify any type of **-through** option, all timing paths specified using the **-from** and **-to** options are affected.

-rise_through ***rise_through_list***

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through ***fall_through_list***

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all sequential elements and primary outputs related to that clock are used as path endpoints. If a cell is specified, all input pins and bidirectional pins of that cell are affected. You can use at most one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge signals to the specified objects are affected. If a

clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edges of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to *fall_to_list*

Same as the **-rise_to** option, except that only falling-edge signals to the specified objects are affected.

-consecutive

Specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

-use_existing_timing_points

When this option is used, the exception definition must refer only to pins that have been defined as timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **set_false_path** command has been issued.

DESCRIPTION

The **set_false_path** command prevents path tracing and timing analysis for the specified paths and stops those paths from being saved in the path database.

This command defines a path-based timing exception, along with the **set_no_same_phase_path**, **set_same_phase_path**, **set_multicycle_path**, **set_no_timing_check_path**, and **set_exclude_reference_path** commands. Path-based timing exceptions have priority over object-based timing exceptions, such as those set with the **set_phasing** or **set_timing_check_attributes** commands or invoked via the **timing_default_phasing** variable.

In case of conflicting exceptions for a path, the timing exception types have the following order of priority, from highest to lowest:

```
1. set_false_path
2. set_no_timing_check_path
3. set_multicycle_path
4. set_no_same_phase_path
5. set_same_phase_path
```

Multiple exception settings that are not in conflict with each other can apply to a single path. There is no conflict between the **set_multicycle_path** command and the **set_same_phase_path** or **set_no_same_phase_path** commands. Also, there is no conflict between the **-setup** and **-hold** settings for a specific exception.

To undo the effects of a **set_false_path** command, use the **remove_false_path** command.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any one of the listed objects. The **-consecutive** option specifies that the "from," "through," and "to" points must be

consecutive, without other points in between.

To prevent timing checks while still allowing path tracing through specified points in the design, use the **set_no_timing_check_path** command rather than the **set_false_path** command.

Instead of using the **set_false_path** command to exclude paths, you might want to specify a set of paths to analyze and exclude all other paths. To do so, use the **set_find_path** command.

NOTE: The **set_false_path** command is a Synopsys Design Constraints (SDC) command. However, the **-rise_from**, **-rise_to**, **-rise_through**, **-fall_from**, **-fall_to**, and **fall_through** options might not be supported by other tools that use SDC commands. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_false_path** command is typically used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1:

```
nt_shell> set_false_path -from A1 -through B1 -through C1 -to D1
```

The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1:

```
nt_shell> set_false_path -from A1 -through {B1 B2} -through {C1 C2} -to D1
```

A design has two cells, ff12 and ff34, that are never used at the same time. The following commands declare the timing paths between the cells to be false paths. These paths are not traced and not saved in the path database.

```
nt_shell> set_false_path \
          -from [get_cells ff12] \
          -to [get_cells ff34]
nt_shell> set_false_path \
          -from [get_cells ff34] \
          -to [get_cells ff12]
```

The following command disables path tracing for minimum delay analysis for paths with endpoints clocked by CLK1:

```
nt_shell> set_false_path -hold \
          -to [get_clocks CLK1]
```

SEE ALSO

```
remove_false_path(2)
report_exceptions(2)
set_exclude_reference_path(2)
set_multicycle_path(2)
set_no_same_phase_path(2)
set_no_timing_check_path(2)
set_same_phase_path(2)
```

set_fanout_noise_threshold

Sets a new fanout noise threshold on a list of objects (nets, pins).

SYNTAX

```
status set_fanout_noise_threshold
   [-above -low ]
   [-below -high]
   value
   objects
```

Data Types

```
   value       float
   objects     list
```

ARGUMENTS

-above

Valid only when used with the **-low** option to specify the threshold for fanout noise values going above the ground rail.

-low

Valid only when used with the **-above** option to specify the threshold for fanout noise values going above the ground rail.

-below

Valid only when used with the **-high** option to specify the threshold for fanout noise values going below the supply rail.

-high

Valid only when used with the **-below** option to specify the threshold for fanout noise values going below the supply rail.

value

A positive floating-point value that specifies the new threshold value.

objects

Specifies the objects in the design to which a new fanout noise threshold apply. Any net specified maps to the set of leaf pins connected to that net.

DESCRIPTION

This command is used to specify pin-specific fanout noise propagation thresholds and override the global thresholds.

By default, the **si_fanout_noise_threshold_above_low** and **si_fanout_noise_threshold_below_high** variables are used to determine if a noise waveform should be propagated to its fanout nets. If the noise peak on a victim exceeds these global threshold values, the effect of this waveform is propagated to each of the nets it drives.

The **set_fanout_noise_threshold** command can be used to replace one or both of the global thresholds for a specific set of pins in the design. The threshold must be a positive value which is added to the ground voltage or subtracted from the supply voltage to determine whether noise waveforms on the objects in the list should be propagated for fanout noise analysis.

The **set_fanout_noise_threshold** command has an effect only when used with the **-above -low** or **-below -high** combinations of options. Other combinations have no effect, but do not generate a warning message.

COMMAND PHASING

The **set_fanout_noise_threshold** command can be executed any time after the **check_design** command. It must be executed before the **update_noise** command to affect the noise analysis.

EXAMPLES

The following command sets a fanout noise threshold of 0.1 volts on pin Xxor7.MP1.g.

```
nt_shell> set_fanout_noise_threshold 0.1 Xxor7.MP1.g
```

The following command sets a fanout noise threshold of 0.2 volts for peaks above the ground rail on pin Xxor2.MN1.g.

```
nt_shell> set_fanout_noise_threshold -above -low 0.2 Xxor2.MN1.g
```

SEE ALSO

```
report_noise(2)
report_fanout_noise(2)
si_enable_noise_analysis(3)
si_enable_noise_fanout_analysis(3)
si_fanout_noise_threshold_above_low(3)
si_fanout_noise_threshold_below_high(3)
```

set_find_path

Specifies the paths in the design to be traced by the **trace_paths** command based on startpoints, throughpoints, endpoints, and edge types.

SYNTAX

```
status set_find_path
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-consecutive]
   [-use_existing_timing_points]
   [-max]
   [-min]
```

Data Types

```
   from_list            list
   rise_from_list       list
   fall_from_list       list
   to_list              list
   rise_to_list         list
   fall_to_list         list
   through_list         list
   rise_through_list    list
   fall_through_list    list
```

ARGUMENTS

-from *from_list*

Specifies a list of pins, ports, or nets to be considered path startpoints, from which path tracing starts. Path startpoints are typically the input ports and clock pins of sequential devices.

-rise_from *rise_from_list*

Same as the **-from** option, except that only rising-edge transitions from the specified objects are

considered.

-fall_from ***fall_from_list***

Same as the **-from** option, except that only falling-edge transitions from the specified objects are considered.

-to ***to_list***

Specifies a list of pins, ports, or nets to be considered path endpoints, where path tracing stops. Path endpoints are typically the output ports and data pins of sequential devices.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_to ***fall_to_list***

Same as the **-to** option, except that only falling-edge transitions arriving at the specified objects are considered.

-through ***through_list***

Specifies a list of pins, ports, or nets through which the paths must pass to be considered. If you specify multiple objects in a *through_list* for a single **-through** option, a path can pass through any one of those objects. If you use multiple **-through** options, a path must pass through each **-through** specification in sequential order.

-rise_through ***rise_through_list***

Same as the **-through** option, except that only rising-edge transitions arriving at the specified objects are considered.

-fall_through ***fall_through_list***

Same as the **-through** option, except that only falling-edge transitions arriving at the specified objects are considered.

-consecutive

Matches a path only if the specified points along the path are consecutive.

-use_existing_timing_points

When this option is used, the exception definition must only refer to pins which are already timing points. This can only be true after the **check_design** has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** does not need to be run again after the **set_find_path** command has been issued.

-max

Restricts path tracing to maximum-delay paths.

-min

Restricts path tracing to minimum-delay paths.

DESCRIPTION

The **set_find_path** command restricts the scope of path tracing to paths that start at, pass through, or end at specified ports, pins, or nets. The command specifies the paths in the design to be checked by the **check_design** command and traced by the **trace_paths** command. Only the paths that meet the **set_find_path** specifications are traced. Restricting the paths with **set_find_path** reduces runtime and memory usage.

You cannot use the **-through**, **-rise_through**, and **-fall_through** options with the **set_find_path** command if you specify clock objects with the **-from** or **-to** or related options.

You can use the **set_find_path** multiple times to specify multiple sets of paths to be traced. NanoTime processes each set of paths sequentially.

If you use different options within the same command, a path must qualify for all of those options to be traced. If you specify a list of objects for a single option, a path only needs to qualify for any one of the objects in the list to be traced. If multiple **-through** options are used, a path must pass through the objects in the order specified in the command. For example:

```
nt_shell> set_find_path -from {A B} \
          -through {C D} -through {E} -to {F}
```

In this example, a path is traced only if it starts from either A or B, then passes through either C or D, then passes through E, and then ends at F.

You can report the path restrictions defined by **set_find_path** commands by using the **report_find_path** command.

To cancel the effects of the **set_find_path** command, use the **remove_find_path** command.

COMMAND PHASING

The **set_find_path** command is typically used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option with the **set_find_path** command, you can execute it after the **check_design** command and before the **trace_paths** command or **extract_model** command.

EXAMPLES

The following command selects all paths that start with a rising edge on input rst.

```
nt_shell> set_find_path -rise_from [get_ports rst]
```

The following command selects the paths that start from port IN1 and end on either port OUT1 or port

OUT2.

```
nt_shell> set_find_path \
          -from [get_ports IN1] \
          -to [get_ports {OUT1 OUT2}]
```

The following command selects any maximum-delay path that starts at either port A1 or port A2, then passes through either pin X1/Q or X2/Q, then passes through pin X8/Q, and then ends at port D4.

```
nt_shell> set_find_path \
          -from [get_ports {A1 A2}] \
          -through [get_pins {X1/Q X2/Q} \
          -through [get_pins {X8/Q} \
          -to [get_ports D4] -max
```

SEE ALSO

check_design(2)
remove_find_path(2)
report_find_path(2)
trace_paths(2)

set_gated_clock_timing_check_attributes

Modifies a gated clock check created by the **create_gated_clock_timing_check** command.

SYNTAX

```
status  set_gated_clock_timing_check_attributes
   {-from from_object
      | -rise_from from_object
      | -fall_from from_object}
   {-to to_object
      | -rise_to to_object
      | -fall_to to_object}
   [-setup | -hold]
   check_value
```

Data Types

```
   from_object          list
   to_object            list
   check_value          float
```

ARGUMENTS

-from *from_object*

Specifies a pin or port in the current design as the clock signal being gated. You must specify one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising transitions at the clock pin.

-fall_from *from_object*

Similar to the **-from** option, but applies only to falling transitions at the clock pin.

-to *to_object*

Specifies a pin or port in the current design as the clock-gating signal. You must specify one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to *to_object*

Similar to the **-to** option, but applies only to rising signals at the clock-gating pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling signals at the clock-gating pin.

-setup

Indicates that the data check value is for setup analysis only.

-hold

Indicates that the data check value is for hold analysis only. If neither of the **-setup** or **-hold** options is specified, the value applies to both setup and hold.

check_value

Specifies the value of the setup or hold time for the check.

DESCRIPTION

NanoTime performs clock gating setup and hold checks at clock gates that it recognizes. If there is a clock gate that NanoTime does not recognize, you can manually specify a clock gating check for that clock gating structure with the **create_gated_clock_timing_check** command.

To modify an existing gated clock check, use the **set_gated_clock_timing_check_attributes** command. The command options are the same as for the **create_gated_clock_timing_check** command, and work in the same manner. The modification should be done after the **check_design** command, but before the **trace_paths** command.

The "from" and "to" specifications must match the ones used in the original **create_gated_clock_timing_check** command, as well as the **-setup** and **-hold** options, if used originally. If there is no match with a previous **set_gated_clock_check** command, a warning is generated and the command has no effect on the design.

To remove (rather than modify) a timing check set with the **create_gated_clock_timing_check** command, use the **remove_gated_clock_check** command.

COMMAND PHASING

The **set_gated_clock_timing_check_attributes** command is typically executed in the "design-checked" state (in other words, after the **check_design** command but before any path tracing commands (such as the **trace_paths** or **extract_model** commands). If you use the **set_gated_clock_timing_check_attributes** command after the **check_design** command, NanoTime transitions back to the "design-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

There following commands assign new setup and hold times to existing checks previously set with the **create_gated_clock_timing_check** command. The "from" and "to" specifications must exactly match the ones used in the original **create_gated_clock_timing_check** commands.

```
nt_shell> set_gated_clock_timing_check_attributes \
          -rise_from [get_ports CLK] \
          -to [get_ports CLKGATE] -setup 0.4

nt_shell> set_gated_clock_timing_check_attributes \
          -rise_from [get_ports CLK] \
          -to [get_ports CLKGATE] -hold 0.6
```

SEE ALSO

create_gated_clock_timing_check(2)
remove_gated_clock_check(2)
report_paths(2)
trace_paths(2)
timing_clock_gate_hold_fall_margin(3)
timing_clock_gate_hold_rise_margin(3)
timing_clock_gate_setup_fall_margin(3)
timing_clock_gate_setup_rise_margin(3)

set_hierarchy_separator

Sets the hierarchy separator character.

SYNTAX

```
status set_hierarchy_separator
   hierarchy_separator
```

Data Types

```
   hierarchy_separator          string
```

ARGUMENTS

hierarchy_separator

The hierarchy separator character, one of @ ^ # . / |

DESCRIPTION

This command specifies the ASCII character that is to be used to delimit hierarchical levels when NanoTime reads in a netlist and when the tool generates reports with hierarchical names. By default, the 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 also specify the hierarchy separator character by setting the **hierarchy_separator** variable. Using the command and setting the variable have the same effect. For example, these two commands are the same:

```
nt_shell> set_hierarchy_separator |
1
nt_shell> set hierarchy_separator |
|
```

To view the current setting, use the following command:

```
nt_shell> printvar hierarchy_separator
hierarchy_separator  = "|"
```

In most situations, you should use the default separator, which is the period character. However, in some 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 way to deal with this situation.

COMMAND PHASING

The **set_hierarchy_separator** command is not phase-restricted.

EXAMPLES

Consider a design that contains a hierarchical cell A, which contains hierarchical cells B and B.C; B.C contains D; B contains C. Searching for "A.B.C.D" is ambiguous and might not match what you want. However, if you set the hierarchy separator character to the vertical bar, searching for "A | B.C | D" or for "A | B | C | D" is clear and unambiguous.

```
nt_shell> set_hierarchy_separator |
nt_shell> get_cell
```

SEE ALSO

```
register_netlist(2)
link_design(2)
hierarchy_separator(3)
```

set_hspice_timing_model_paths

Specifies the paths to be modeled with HSPICE timing when the **extract_model -hspice_timing** command is executed.

SYNTAX

```
status set_hspice_timing_model_paths
   [-from from_list]
   [-to to_list]
   [-through through_list]
   [-max]
   [-min]
   [-path_delay_upper_threshold delay_value]
   [-num_arcs_upper_threshold number]
```

Data Types

```
from_list      list
to_list        list
through_list   list
delay_value    float
number         integer
```

ARGUMENTS

-from *from_list*

Specifies a list of ports as path startpoints (typically the input ports).

-to *to_list*

Specifies a list of ports to be considered path endpoints, where path tracing stops. Path endpoints are typically the output ports.

-through *through_list*

Specifies a list of transistor gate pins through which the paths must pass in order to be modeled by HSPICE. If you specify multiple pins in a *through_list* for a single **-through** option, a path can pass through any one of those objects. If you use multiple **-through** options, a path must pass through each **-through** specification in sequential order.

-max

Restricts path tracing to maximum-delay paths.

-min

Restricts path tracing to minimum-delay paths.

-path_delay_upper_threshold *delay_value*

Specifies the maximum delay that a path can have to be selected.

-num_arcs_upper_threshold *arc_number*

Specifies the maximum number of arcs that a path can have to be selected.

DESCRIPTION

By default, the **extract_model -hspice_timing** command uses HSPICE to analyze the complete set of paths included in the timing model. To reduce runtime, you can optionally use the **set_hspice_timing_model_paths** command to specify the paths in the design to be modeled with HSPICE. You can use the command multiple times.

You can specify the path startpoints, throughpoints, endpoints, and edge types. Only the paths that meet the delay and arc specifications are modeled using HSPICE. The remaining paths are modeled using the NanoTime internal simulator.

To reverse the effects of the **set_hspice_timing_model_paths** command, use the **remove_hspice_timing_model_paths** command.

COMMAND PHASING

Use the **set_hspice_timing_model_paths** command after the **check_design** command and before the **extract_model** command.

SEE ALSO

extract_model(2)
remove_hspice_timing_model_paths(2)
trace_paths(2)

set_initial_condition_adjustment

Adjusts initial conditions within the channel-connected blocks of specified nets.

SYNTAX

```
status set_initial_condition_adjustment
   [-nfactor n_factor]
   [-pfactor p_factor]
   net_list
```

Data Types

```
   n_factor     float
   p_factor     float
   net_list     list
```

ARGUMENTS

-nfactor *n_factor*

The threshold voltage factor for NMOS devices. NanoTime subtracts the threshold voltage times this factor from the supply voltage to obtain the weak-Vdd initial condition. The default is 1.0.

-pfactor *p_factor*

The threshold voltage factor for PMOS devices. NanoTime adds the threshold voltage times this factor to the ground voltage to obtain the weak-Gnd initial condition. The default is 1.0.

net_list

The list of nets. The channel-connected blocks of the specified nets are affected by the command.

DESCRIPTION

This command adjusts the initial conditions within the channel-connected blocks of specified nets. NanoTime sets equilibrium initial conditions within a channel-connected block when building a simulation

unit. Depending on the input vector and input/output transitions, the equilibrium initial condition of a net could be Vdd, Gnd, weak-Vdd, or weak-Gnd, where

```
weak-Vdd = Vdd -  Vth,n
weak-Gnd = Gnd + |Vth,p|
```

This command adjusts the initial conditions of weak-Vdd and weak-Gnd as follows:

```
weak-Vdd = Vdd - nfactor *  Vth,n
weak-Gnd = Gnd + pfactor * |Vth,p|
```

To remove initial condition adjustments that have been set, use the **remove_initial_condition_adjustment** command. To limit the allowable initial condition voltages, use the **weak_gnd_ic_multiplier_limit** and **weak_vdd_ic_multiplier_limit** variables. To use a current-based method for calculating initial conditions in a FinFET design, set the **tech_use_current_based_vth_analysis** variable to **true**.

COMMAND PHASING

The **set_initial_condition_adjustment** command can be used any time after the **link_design** and before the **extract_model** or **trace_paths** command. If you use the **set_initial_condition_adjustment** command after the **extract_model** or **trace_paths** command, NanoTime transitions back to the "design-checked" state and discards all commands and operations performed after the **extract_model** or **trace_paths** command.

EXAMPLES

The following example adjusts the initial condition of the net within the NAND gate NMOS stack by a factor of -1. After adjustment, the initial condition of the node is Vdd + Vth,n

```
nt_shell> set_initial_condition_adjustment \
          -nfactor -1 [get_nets nand.out]
```

SEE ALSO

```
remove_initial_condition_adjustment(2)
weak_gnd_ic_multiplier_limit(3)
weak_vdd_ic_multiplier_limit(3)
tech_use_current_based_vth_analysis(3)
```

set_input_delay

Specifies the data arrival time at an input relative to a clock.

SYNTAX

```
status set_input_delay
   [-clock clock_name]
   [-clock_fall]
   [-level_sensitive]
   [-asynchronous]
   [-rise]
   [-fall]
   [-max]
   [-min]
   [-add_delay]
   [-network_latency_included]
   [-source_latency_included]
   delay_value
   port_pin_list
```

Data Types

```
   clock_name            list
   delay_value           float
   port_pin_list         list
```

ARGUMENTS

-clock *clock_name*

Specifies the clock to which the specified delay is related. If you do not use the **-clock** option, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with a period determined by considering the sequential elements in the transitive fanout of each port.

-clock_fall

Specifies that the delay is relative to the falling edge of the clock specified by the **-clock** option, rather than the rising edge of that clock.

-level_sensitive

Specifies that the source of the delay is a level-sensitive latch. This causes NanoTime to assume that the input delay applies to data launched on the closing edge of the related clock, rather than the opening edge.

-asynchronous

Specifies that the input is not tied to a clock. This option is required to activate recovery or removal timing checks, and consequently disqualifies synchronous timing checks from being evaluated (this includes any latch, flop, precharge, and/or gated clock setup and hold checks whose constraint pin is driven by a path from this asynchronous input). The **-clock** option cannot be used together with this option.

-rise

Specifies that the *delay_value* argument refers to a rising transition on specified ports of the current design. If you do not use the **-rise** or **-fall** options, rising and falling delays are assumed to be equal.

-fall

Specifies that the *delay_value* argument refers to a falling transition on specified ports of the current design.

-max

Specifies that the *delay_value* argument refers to the longest path. If you do not specify the **-max** or **-min** options, maximum and minimum input delays are assumed to be equal.

-min

Specifies that the *delay_value* argument refers to the shortest path.

-add_delay

Adds the delay information to the existing input delay, instead of overwriting it. Use the **-add_delay** option to capture information about multiple paths leading to an input port that are relative to different clocks or different edges of the same clock.

-network_latency_included

Specifies whether the clock network latency should be included in the input delay value for an ideal clock. If this option is not specified, the clock network latency of the related clock is added to the input delay value. The option has no effect if the clock is propagated or the input delay is not specified with respect to any clock.

-source_latency_included

Specifies whether to include the clock source latency in the input delay value. If this option is not specified , the clock source latency of the related clock is added to the input delay value. The option has no effect if the input delay is not specified with respect to any clock.

delay_value

Specifies the amount of time that passes after a clock edge before a signal becomes available at an input, in time units of the technology. This usually represents a combinational path delay from an external sequential element to the input.

port_pin_list

Specifies the names of input ports or internal pins in the current design to which the *delay_value*

argument is assigned. If you specify more than one object, the objects must be enclosed in braces {}.

DESCRIPTION

This command sets input path delay values for the current design. The input delay is the amount of time it takes for a signal to arrive at the input after the clock edge that launches the data toward the input. If you do not specify an input delay for a port, NanoTime assumes a delay of zero. The input and output delays characterize the operating environment of the current design, together with the **set_load** and **set_drive** commands.

Use the **-rise** or **-fall** option to restrict the input delay setting to only rising or only falling edges at the specified ports. Use the **-min** or **-max** option to specify different worst-case minimum and maximum input delays.

The **set_input_delay** command sets the amount of delay from an external launch element to the input port relative to a clock edge. The clock is specified with the **-clock** option. Use the **-clock_fall** option if the delay is relative to the falling edge rather than the rising edge of the clock.

If the external launch element is a domino precharge gate, use the **-level_sensitive** option. In that case, when NanoTime performs setup and hold checking, it assumes that the specified input delay applies to the launch of data on the closing edge of the clock, rather than the opening edge.

You can use the **-add_delay** option to specify input delays to a port relative to different clocks, or different edges of the same clock (with and without the **-clock_fall** option). First use a **set_input_delay** command to specify the delay with respect to the one clock, then use additional **set_input_delay** commands with the **-add_delay** option to specify the delays relative to other clocks. In this case NanoTime stores the delay information separately for each clock edge, instead of overwriting existing delay values.

To report input delays of internal pins, use the **report_paths** command.

To remove input delay values, use the **remove_input_delay** or **reset_design** commands.

NOTE: The **set_input_delay** command is a Synopsys Design Constraints (SDC) command. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_input_delay** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_input_delay** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example sets an input delay of 2.3 for ports IN1 and IN2 on a combinational design. Because the design is combinational, no clock is needed.

```
nt_shell> set_input_delay 2.3 {IN1 IN2}
```

The following example sets an input delay of 1.2 relative to the rising edge of CLK1 for all input ports in the design.

```
nt_shell> set_input_delay 1.2 -clock CLK1 \
          [all_inputs]
```

The following example sets the input and output delays for the bidirectional port INOUT1. The input signal arrives at INOUT1 2.5 units after the falling edge of CLK1. The output signal is required at INOUT1 at 1.4 units before the rising edge of CLK2.

```
nt_shell> set_input_delay 2.5 -clock CLK1 \
          -clock_fall [get_ports INOUT1]
nt_shell> set_output_delay 1.4 -clock CLK2 \
          [get_ports INOUT1]
```

The following example models the situation where there are three paths to input port IN1. The first path is relative to the rising edge of CLK1. The second path is relative to the falling edge of CLK1. The third path is relative to the falling edge of CLK2. The **-add_delay** option adds the new input delay information without overwriting the old information.

```
nt_shell> set_input_delay 2.2 -max clock CLK1 \
          -add_delay {IN1}
nt_shell> set_input_delay 1.7 -max clock CLK1 \
          -clock_fall -add_delay {IN1}
nt_shell> set_input_delay 4.3 -max clock CLK2 \
          -clock_fall -add_delay {IN1}

To get a report on input delays that have been set, use the report_port -input_delay
command. For example:
nt_shell> set_input_delay -clock MCLK -clock_fall \
          -min 0.31 [get_ports BUS[0]]
1
nt_shell> set_input_delay -clock MCLK -clock_fall \
          -max 0.32 [get_ports BUS[0]]
1
nt_shell> report_port -input_delay BUS[0]
 ...
Input Delay

Input Port Min Rise Min Fall Max Rise Max Fall Related Clock
---------- -------- -------- -------- -------- -------------
BUS[0]        0.310    0.310    0.320    0.320 MCLK(f)
```

SEE ALSO

```
all_inputs(2)
create_clock(2)
current_design(2)
remove_input_delay(2)
report_design(2)
report_port(2)
reset_design(2)
set_drive(2)
set_load(2)
set_output_delay(2)
```

set_input_noise

Sets a user-defined injected noise bump on a list of objects (ports or pins).

SYNTAX

```
status set_input_noise
   [-add_noise]
   [-above]
   [-below]
   [-high]
   [-low]
   -height height_value
   -width width_value
   [-time_to_peak_ratio ratio_value]
   objects
```

Data Types

```
   height_value   float
   width_value    float
   ratio_value    float
   objects        list
```

ARGUMENTS

-add_noise

Specifies a noise bump that is added to existing calculated injected noise. If not specified, the calculated injected noise is overridden.

-above

Specifies a noise bump that goes above the quiet value of the victim net.

-below

Specifies a noise bump that goes below the quiet value of the victim net.

-high

Specifies a noise bump that occurs with the victim net held at Vdd.

-low

Specifies a noise bump that occurs with the victim net held at ground.

-height

Specifies the height of the noise bump.

-width

Specifies the width of the noise bump.

-time_to_peak_ratio

Specifies the time-to-peak ratio of the noise bump. If not specified, a value of 0.5 (symmetrical noise bump) is assumed.

objects

Specifies the objects in the design to which the user-defined noise applies.

DESCRIPTION

This command is used to specify a user-defined injected noise bump on a selected set of pins. The **-above, -below, -high,** and **-low** options can be used in combination to be more specific.

COMMAND PHASING The **set_input_noise** command can only be executed after path tracing is complete.

EXAMPLES

The following command sets an above high noise bump of 0.3 volts, 0.3 units of time, and a 0.3 time-to-peak ratio on pin Xxor7.MP1.g:

```
nt_shell> set_input_noise -above -high -height 0.3 -width 0.3 \
          -time_to_peak_ratio 0.3 Xxor7.MP1.g
```

SEE ALSO

report_noise(2)
set_input_noise(2)
report_fanout_noise(2)
si_enable_noise_analysis(3)

set_input_transition

Sets a fixed transition time on input ports or bidirectional ports.

SYNTAX

```
status set_input_transition
   [-rise]
   [-fall]
   [-min]
   [-max]
   [-clock clock_name]
   [-clock_fall]
   [-rail_voltage volts]
   transition
   port_list
```

Data Types

clock_name	string
volts	float
transition	float
port_list	list

ARGUMENTS

-rise

Sets only the rising transition time.

-fall

Sets only the falling transition time.

-min

Sets only the worst-case minimum transition time.

-max

Sets only the worst-case maximum transition time.

-clock *clock_name*

The input transition applies only to external paths driven by the specified clock.

-clock_fall

The input transition applies only to external paths driven by falling edges of the clock. The default is the rising edge.

-rail_voltage ***volts***

The rail voltage at the input. NanoTime uses the rail voltage to generate an accurate waveform for simulation of the first stage of a path that begins at the input. In the absence of this option, NanoTime uses the **oc_global_voltage** setting.

transition

The transition time at the port. This is a floating-point number greater than or equal to 0.0.

port_list

A list of input ports or bidirectional ports.

DESCRIPTION

The transition time of a signal (also known as slew) is the amount of time it takes for the signal to change from one logic state to the other.

The **set_input_transition** command specifies a fixed transition time for a list of input ports or bidirectional ports. This transition time affects the calculation of delays for nets and cells in the transitive fanout of the port.

The transition time can be specified relative to a clock by using the **-clock** option. This means that the transition time applies only to external paths driven by the specified clock. For a falling-edge clock, use the **-clock_fall** option together with the **-clock** option.

To view the transition times that have been set on ports, use the **report_port -drive** command.

If no transition time is set for a port, NanoTime uses the transition times determined by the following variables:

```
sim_transition_max_fall
sim_transition_max_rise
sim_transition_min_fall
sim_transition_min_rise
```

These variables are all set by default to 0.05 nanoseconds.

The **set_drive** command models the external driver at an input port as a voltage source and a linear resistance. If you use both the **set_drive** and **set_input_transition** commands, NanoTime uses the specified transition time at the ideal driver and also considers the port load to determine the final effective transition time for the port.

NOTE: The **set_input_transition** command is a Synopsys Design Constraints (SDC) command. However, the **-rail_voltage** option might not be supported by other tools that use SDC commands. SDC

command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_input_transition** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_input_transition** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

This example specifies that ports matching the pattern DATA_IN* have a transition time of 0.75 units.

```
nt_shell> set_input_transition 0.75 \
          [get_ports DATA_IN*]
```

SEE ALSO

set_drive(2)
report_port(2)
set_clock_transition(2)
oc_global_voltage(3)
sim_transition_max_fall(3)
sim_transition_max_rise(3)
sim_transition_min_fall(3)
sim_transition_min_rise(3)

set_libcell_variation_parameters

Sets the variation value to be used for all delay arcs of a specific set of libcells.

SYNTAX

```
status set_libcell_variation_parameters
   -variation value
   [-min | -max]
   libcells
```

Data Types

```
   value            float
   libcells         list
```

ARGUMENTS

-variation *value*

The **set_libcell_variation_parameters** command specifies the amount of variation to apply for all arc delays through a list of libcells. The nominal delay is computed for each delay in the path, then the variation amount is calculated by multiplying the nominal delay by the variation value specified in this command. The calculated value is used when determining the full path variation value. The calculated value represents a one sigma variation value.

-min

Only applies during minimum delay path tracing and delay calculation.

-max

Only applies during maximum delay path tracing and delay calculation.

libcells

A list of libcells to which this variation value applies.

DESCRIPTION

The **set_libcell_variation_parameters** command is used to specify how variation for arc delays of a list of libcells is computed. These variation values are used in the context of a full path and are combined in a statistical fashion to compute an overall variation adjustment for the path.

If a value for a specific libcell is defined with the **set_libcell_variation_parameters** command, that value takes precedence over a LVF (Library Variation Format) table in a library file or a value defined by the **set_variation_parameters** command using the type "libcell" or "default".

SEE ALSO

```
set_variation_parameters(2)
report_variation(2)
timing_pocv_sigma(3)
```

set_load

Sets the load capacitance on specified ports or nets in the design.

SYNTAX

```
status set_load
   [-add]
   [-min]
   [-max]
   [-rise]
   [-fall]
   [-pin_load]
   [-wire_load]
   value
   objects
```

Data Types

value	float
objects	list

ARGUMENTS

-add

Adds the specified capacitance value to the existing capacitance on the port or net, instead of replacing it.

-min

Specifies the worst-case minimum delay analysis capacitance.

-max

Specifies the worst-case maximum delay analysis capacitance.

-rise

Specifies that the capacitance applies to rising transitions only.

-fall

Specifies that the capacitance applies to falling transitions only.

-pin_load

Specifies the pin capacitance of a port. (This option should not be used to set the load on a net.)

-wire_load

Specifies the wire capacitance of a port. (This option should not be used to set the load on a net.)

value

Specifies the capacitance value in the units of the technology.

objects

A list of ports or nets in the current design on which the capacitance value is set. Use the **get_ports** or **get_nets** command to specify the targeted objects. Otherwise, NanoTime first searches for ports, then nets, that match the specified names.

DESCRIPTION

This command sets the load capacitance of specified ports or nets in the current design. In the command, you must specify the capacitance (in units of the technology) and the ports or nets in the design to which that capacitance value applies. The **-add** option adds the specified value to the existing capacitance value. Otherwise, the specified value replaces the existing value and overrides the internally estimated net capacitance.

By default, the **set_load** command does not affect any net that has been back-annotated by the **read_parasitics** command. Attempting to use the **set_load** command in this case results in a warning message, with no change to the net. To allow the **set_load** command to override any back-annotated capacitance, set the **parasitics_allow_spf_net_override** variable to **true**.

When you apply the load value to a port, you can use the **-pin_load** and **-wire_load** options to specify whether the load value is the pin capacitance or wire capacitance of the port. If neither option is used and the specified object is a port, the specified capacitance is assigned to the pin capacitance.

Use the **-min** or **-max** options to restrict the load value to minimum or maximum delay analysis. If neither is specified, the same value applies to both conditions. Use the **-rise** or **-fall** options to apply the load value to rising or falling edges only. Otherwise, the value applies to both rising and falling edges.

Each port or net can have different values for wire capacitance and pin capacitance; for rising and falling transitions; for minimum delay and maximum delay analysis; and for datapaths and clock paths. Thus, a port or net can have up to 16 different capacitance values. The total capacitance of a net is the sum of the pin, port, and wire capacitances associated with that net. These values can vary with the edge type, min-max condition, and path type.

To view the capacitance values on a port, use the **report_port** command. To view the capacitance values on a net, use the **report_net -connections** command. To view the capacitance attributes of a port or net, use the **report_attribute** command.

To remove capacitance values that have been set on a port or net, use the **remove_capacitance**

command. To reset all annotated capacitance values in a design, use the **reset_design** command.

NOTE: The **set_load** command is a Synopsys Design Constraints (SDC) command. However, the **add**, **-rise**, and **-fall** options might not be supported by other tools that use SDC commands. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_load** command is typically executed in the "netlist-linked" or "topology-checked" state (in other words, between the **link_design** and **check_design** commands). You cannot use the command before the **link_design** command. If you use the **set_load** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example sets a capacitance of 2.5 units on the port named out1.

```
nt_shell> set_load 2.5 [get_ports out1]
1
nt_shell> report_port out1
 ...
              Min Rise Min Fall Max Rise Max Fall
Port    Dir         Cap      Cap      Cap      Cap
------- ----- -------- -------- -------- --------
ox1     inout    2.500    2.500    2.500    2.500
```

The following example uses the **get_attribute** command to get the value of the **pin_capacitance_fall_max** attribute on pin X5.Mn0.d and sets that value on ports in1 and in2.

```
nt_shell> set_load [get_attribute \
                [get_lib_pins X5.Mn0.d] pin_capacitance_fall_max] \
                [get_ports {in1 in2}]
```

The following example sets a wire capacitance of 3 on the net U1.U2.NET3. The total net capacitance is 3 plus the sum of the pin and port capacitance values.

```
nt_shell> set_load 3 U1.U2.NET3
```

The following example sets a wire capacitance of 5 units on the port named ox6.

```
nt_shell> set_load -wire_load 5 [get_ports ox6]
```

The following example removes the capacitance on port ox6 and net n12.

```
nt_shell> remove_capacitance [get_ports ox6]
nt_shell> remove_capacitance [get_nets n12]
```

SEE ALSO

```
read_parasitics(2)
remove_capacitance(2)
report_net(2)
report_attribute(2)
report_port(2)
reset_design(2)
parasitics_allow_spf_net_override(3)
```

set_logic_constraint

Sets logic constraints on specified pins or nets.

SYNTAX

```
status set_logic_constraint
   -one_hot | -one_off | -at_most_one_hot | -at_most_one_off
   | -invert | -equal | -nand | -nor
   [-crosstalk_force]
   pins_or_nets
```

Data Types

```
   pins_or_nets         list
```

ARGUMENTS

-one_hot

At any given time, exactly one of the nets in the specified list has a value of logic 1; the others have a value of logic 0.

-one_off

At any given time, exactly one of the nets in the specified list has a value of logic 0; the others have a value of logic 1.

-at_most_one_hot

At any given time, no more than one of the nets in the specified list can have a value of logic 1.

-at_most_one_off

At any given time, no more than one of the nets in the specified list can have a value of logic 0.

-invert

The first specified pin or net is the input of an inverter; the second is the output of the inverter. Exactly two objects must be specified.

-equal

At any given time, all of the listed nets have the same value, either logic 0 or logic 1.

-nand

The last specified pin or net is the output of a logical NAND; other pins or nets are the inputs of the NAND.

-nor

The last specified pin or net is the output of a logical NOR; other pins or nets are the inputs of the NOR.

-crosstalk_force

Makes the logic constraint have a higher priority in aggressor logic pessimism reduction for SI delay and noise analysis. Logic constraints with this option will be evaluated first during SI delay and noise analysis. Logic constraints without this option will be evaluated after all constraints with this option are evaluated. The option cannot be used with -equal, -nand, or -nor options.

pins_or_nets

The list of related pins or nets to which the logical constraint applies.

DESCRIPTION

The **set_logic_constraint** command sets logic constraints on a specified list of pins or nets, thereby restricting the scope of the analysis to the specified logic conditions.

Setting or removing logic constraints changes the design, making it necessary to run the **check_design** and **trace_paths** commands to get new analysis results.

COMMAND PHASING

The **set_logic_constraint** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_logic_constraint** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command defines the set of control signals ct0 through ct3 to be "one hot." This means that at any given time, exactly one signal is at logic 1 and the other three are at logic 0.

```
nt_shell> set_logic_constraint -one_hot \
          [get_nets {ct0 ct1 ct2 ct3}]
```

SEE ALSO

```
remove_logic_constraint(2)
report_logic_constraint(2)
set_case_analysis(2)
```

set_lvs_equivalent_nets

Declares nets to be equivalent.

SYNTAX

```
status set_lvs_equivalent_nets
   [-force]
   [-nets net_list]
   [-net_combined_report file_name]
```

Data Types

```
   net_list        list
   file_name       string
```

ARGUMENTS

net_list

A list of nets of the top level circuit to be marked as LVS equivalents.

file_name

The name of a file produced by layout versus schematic tool, in which pairs of equivalent nets are identified.

DESCRIPTION

The **set_lvs_equivalent_nets -nets** command declares two or more nets to be equivalent from the point of view of LVS (layout versus schematic). If the layout versus schematic tool treats transistor stacks as parallel even if they have unconnected internal nodes, NanoTime may issue annotation errors when processing parasitics files. Use the **set_lvs_equivalent_nets -nets** command to instruct NanoTime to consider the two distinct nets as equivalent, eliminating the annotation error. The **set_lvs_equivalent_nets -nets** command may be issued inside a **foreach_match** loop.

If the layout versus schematic tool produces a file of nets that it treated as equivalent, use the **set_lvs_equivalent_nets -net_combined_report** command.

NanoTime checks whether the nodes marked as equivalent with the **set_lvs_equivalent_net** command are indeed equivalent when it processes the command. If NanoTime detects a mismatch in topology or connectivity, it signals an error. To override NanoTime's check and force the nets to be marked as equivalent, without regard to the topology of the connected transistors, add the **-force** qualifier. The default behavior is that nets marked as equivalent must pass the checks.

COMMAND PHASING

The **set_lvs_equivalent_nets** command is used after the **link_design** command and before the **check_topology** command. To be effective, it must precede the **read_parasitics** command.

RESTRICTIONS

In order to use the **set_lvs_equivalent_nets** command, you must change the setting of the **link_enable_wrapper_subckt_parasitics** variable from the default, **true,** to explicitly **false.**

Because the merging of two or more nets cannot be easily undone, if you mistakenly mark two nodes as equivalent nets that should not be equivalent, you must reset the design and repeat **link_design**.

EXAMPLES

The following command defines the two nets n1 and n2 to be equivalent nets. Nets n1 and n2 are the internal stack nets of two parallel but distinct stacks.

```
nt_shell> set_lvs_equivalent_nets -nets {n1 n2}
```

The following command defines the two nets n3 and n4 to be equivalent nets, and to override any objections from the connectivity and topology checker. Nets n3 and n4 are internal stack nets, but have different numbers of connected transistors.

```
nt_shell> set_lvs_equivalent_nets -force -nets {n3 n4}
```

SEE ALSO

report_lvs_equivalent_nets(2)

set_max_delay

Specifies a maximum delay for timing paths.

SYNTAX

```
status set_max_delay
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-use_existing_timing_points]
   delay_value
```

Data Types

```
   from_list              list
   rise_from_list         list
   fall_from_list         list
   through_list           list
   rise_through_list      list
   fall_through_list      list
   to_list                list
   rise_to_list           list
   fall_to_list           list
   delay_value            list
```

ARGUMENTS

-rise

Indicates that only rising path delays are to be constrained. If neither the **-rise** nor the **-fall** option is specified, both rising and falling delays are constrained.

-fall

Indicates that only falling path delays are to be constrained. If neither the **-rise** nor the **-fall** option is

specified, both rising and falling delays are constrained.

-from *from_list*

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, or a pin. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from *rise_from_list*

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-fall_from *fall_from_list*

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-through *through_list*

Specifies a list of pins, ports, and nets through which the paths must pass for maximum delay definition. Nets are interpreted to imply the leaf-level driver pins. If you omit the **-through** option, all timing paths specified using the **-from** and **-to** options are affected. You can specify the **-through** option more than one time in a single command invocation.

-rise_through *rise_through_list*

Applies only to paths with a rising transition at the specified objects and is similar to the **-through** option. You can specify the *rise_through_list* option more than one time in a single command invocation.

-fall_through *fall_through_list*

Applies only to paths and is similar to the **-through** option, but with a falling transition at the specified objects. You can specify the **-fall_through** option more than one time in a single command invocation.

-to *to_list*

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a pin, or a net. Nets are interpreted to imply the leaf-level input pins. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-rise_to *rise_to_list*

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-fall_to *fall_to_list*

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of

the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-use_existing_timing_points

When this option is used, the exception definition must only refer to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **set_max_delay** command has been issued.

delay_value

The required maximum delay value for the specified paths. If a path startpoint has an input delay specified, that delay value is added to the path delay.

DESCRIPTION

This command specifies a required maximum delay for timing paths in the current design. The path length for any startpoint to any endpoint must be less than the specified delay value. You must have a NanoTime Ultra license to use this feature.

Individual maximum delay targets are automatically derived from clock waveforms and port input or output delays. For more information, refer to the **create_clock**, **set_input_delay**, and **set_output_delay** command man pages.

The **set_max_delay** command is a point-to-point timing exception command. For example, the command overrides the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include the **set_multicycle_path**, **set_min_delay**, and **set_false_path** commands. A **set_max_delay** or **set_min_delay** command overrides a **set_multicycle_path** command.

The more general commands apply to more than one path. For example, if the **-from** option is used without a **-to** option, many paths would be selected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
-from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
-from A1 -through {B1 B2} -through {C1 C2} -to D1
```

To list the **max_delay**, **min_delay**, **multicycle_path**, and **false_path** information for the design, use the **report_exceptions** command.

To remove information set by the **set_max_delay** command, use the **remove_max_delay** or **reset_design** command.

EXAMPLES

The following command specifies that any delay path to port Y must be less than 10.0 time units.

```
nt_shell> set_max_delay 10.0 -to {Y}
```

The following command specifies that all paths from ff1a or ff1b to ff2e must have delays less than 15.0 units.

```
nt_shell> set_max_delay 15.0 -from {ff1a ff1b} -to {ff2e}
```

The following example specifies that all paths to endpoints clocked by PHI2 must have delays less than 8.5 units.

```
nt_shell> set_max_delay 8.5 -to [get_clocks PHI2]
```

The following example sets a requirement that all paths leading to ports named busA[*] must have delays less than 5.0.

```
nt_shell> set_max_delay 5.0 -to "busA[*]"
```

The following example specifies that all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} must have delays less than 8.0 units.

```
nt_shell> set_max_delay 8.0 -from ff1/CP -through {U1/Z U2/Z}\
          -through {U3/Z U4/C} -to ff2/D
```

The following example specifies that all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall through one or more of {U3/Z U4/C} must have delays less than 8.0 units.

```
nt_shell> set_max_delay 8.0 -from ff1/CP -rise_through {U1/Z U2/Z}\
          -fall_through {U3/Z U4/C} -to ff2/D
```

SEE ALSO

```
create_clock(2)
remove_max_delay(2)
set_min_delay(2)
remove_min_delay(2)
set_input_delay(2)
set_multicycle_path(2)
set_output_delay(2)
```

set_measurement_threshold

Sets the delay and slew measurement thresholds for specified nets. These thresholds affect the determination of data arrival times and slew values.

SYNTAX

```
status set_measurement_threshold
   [-input_threshold_pct_fall percent]
   [-input_threshold_pct_rise percent]
   [-output_threshold_pct_fall percent]
   [-output_threshold_pct_rise percent]
   [-slew_lower_threshold_pct_fall percent]
   [-slew_lower_threshold_pct_rise percent]
   [-slew_upper_threshold_pct_fall percent]
   [-slew_upper_threshold_pct_rise percent]
   [-slew_derate_from_library factor]
   objects
```

Data Types

```
percent       float
factor        float
objects       list
```

ARGUMENTS

-input_threshold_pct_fall *percent*

The percentage threshold that defines the time measurement point for falling transitions at the startpoint of a delay calculation.

-input_threshold_pct_rise *percent*

The percentage threshold that defines the time measurement point for rising transitions at the startpoint of a delay calculation.

-output_threshold_pct_fall *percent*

The percentage threshold that defines the time measurement point for falling transitions at the endpoint of a delay calculation.

-output_threshold_pct_rise *percent*

The percentage threshold that defines the time measurement point for rising transitions at the endpoint of a delay calculation.

-slew_lower_threshold_pct_fall ***percent***

The percentage threshold that defines the time measurement point at the end of a falling transition, used for calculating slew.

-slew_lower_threshold_pct_rise ***percent***

The percentage threshold that defines the time measurement point at the beginning of a rising transition, used for calculating slew.

-slew_upper_threshold_pct_fall ***percent***

The percentage threshold that defines the time measurement point at the beginning of a falling transition, used for calculating slew.

-slew_upper_threshold_pct_rise ***percent***

The percentage threshold that defines the time measurement point at the end of a rising transition, used for calculating slew.

-slew_derate_from_library ***factor***

Specifies the derating factor needed for the transition times in the Synopsys library to match the characterization trip points used for analysis.

objects

Specifies the nets to which the delay and slew measurement parameters apply.

DESCRIPTION

This command sets the delay and slew measurement thresholds for specified nets, overriding the parameter-set thresholds. These thresholds affect the determination of data arrival times and slew values. The delay threshold value assigned by the **set_measurement_threshold** command must be greater than the slew_lower threshold and less than the slew_upper_threshold for the specified object.

The variables in the following list specify the threshold measurement points of signal transitions and optionally derate library slew values by a specified factor. 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
rc_input_threshold_pct_rise	50
rc_output_threshold_pct_fall	50
rc_output_threshold_pct_rise	50
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

```
rc_slew_derate_from_library          1
```

With the default settings, NanoTime calculates each delay 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 considers 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.

To override the parameter-set thresholds for specified nets in the design, use the **set_measurement_threshold** command. NanoTime overrides the threshold parameters specified in the command for the listed nets, and uses the parameter-specified thresholds otherwise.

To cancel the effects of the **set_measurement_threshold** command, use the **remove_measurement_threshold** command.

Command Phasing

The **set_measurement_threshold** command can only be executed between "netlist-linked" and "design-checked" states. If used after "design-checked", NanoTime transitions back to the "design-checked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The following command sets the slew measurement thresholds to 15% and 85% for all nets beginning with the characters "Sn".

```
nt_shell> set_measurement_threshold \
          -slew_lower_threshold_pct_rise 15 \
          -slew_upper_threshold_pct_rise 85 \
          -slew_lower_threshold_pct_fall 15 \
          -slew_upper_threshold_pct_fall 85 \
          [get_nets Sn*]
```

SEE ALSO

```
read_parasitics(2)
remove_measurement_threshold(2)
report_annotated_parasitics(2)
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_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_rise(3)
```

set_memory_analysis_type

Sets the memory analysis type on specified memories in the design.

SYNTAX

```
status set_memory_analysis_type
   [-minmax count]
   [-custom]
   [-incremental]
   memory_list
```

Data Types

```
   count          integer
   memory_list    list
```

ARGUMENTS

-minmax *count*

Selects the *count* longest and *count* shortest estimated delay bit cells for each wordline and bitline. The default *count* value is 1.

-custom

Disables **-minmax** bit cell selection. When you use this option, you use the **set_memory_bit_selected** option to select bit cells.

-incremental

Performs a **-minmax** selection from the currently selected bit cells only. The **-minmax** option is required with this option.

memory_list

A list of memories to set the analysis type attribute. Each element in this list is either a collection of memories or a pattern matching the memory names.

DESCRIPTION

NanoTime for memories supports automatic and manual selection of bit cells for analysis. By default, the **-minmax** algorithm with an argument of 1 is used to select 1 minimum and maximum bit cell per wordline and bitline. The **-minmax N** option can be used to increase this number to *count* minimum and maximum bit cells per wordline and bitline.

NanoTime for memories also allows custom selection of bit cells for analysis. To do this, use the **-custom** option and then use the **set_memory_bit_selected** and **remove_memory_bit_selected** commands to create the set of selected bit cells. Additionally, if a custom set has been selected, you can modify the list to show a specified number *count* of minimum and maximum bit cells per wordline and bitline by using the **set_memory_analysis_type** with the **-minmax** *count* and **-incremental** options.

COMMAND PHASING

The **set_memory_analysis_type** command can only be executed in the "topology-checked" state.

EXAMPLES

This command sets analysis to one min bit cell and one max bit cell on all memories.

```
nt_shell> set_memory_analysis_type -minmax 1 [get_memory *]
```

SEE ALSO

set_memory_bit_selected(2)
remove_memory_bit_selected(2)
get_memory(2)

set_memory_bit_selected

Selects the bit cells for memory analysis.

SYNTAX

```
status set_memory_bit_selected
   topology_list
```

Data Types

```
   topology_list     list
```

ARGUMENTS

topology_list

A list of ram topology objects to enable for memory analysis.

DESCRIPTION

NanoTime for memories allows custom selection of bit cells for analysis. To do this, use the **set_memory_analysis_type** command with the **-custom** option and then use the **set_memory_bit_selected** and **remove_memory_bit_selected** commands to create the set of selected bit cells.

The bit cells are designated by the ram topologies in the *topology_list*. A collection of ram topologies is created using the **get_topology** command. After execution, the ram topology attribute *is_selected_bit* is set to true.

Use the **remove_memory_bit_selected** command to deselect a ram topology bit cell. After execution, the ram topology attribute *is_selected_bit* is set to false.

COMMAND PHASING

The **set_memory_bit_selected** command can only be executed in the "topology-checked" state.

EXAMPLES

This command selects all the ram topologies in the collection for memory analysis.

```
nt_shell> set_memory_bit_selected [get_topology -structure_type ram *]
```

SEE ALSO

get_topology(2)
remove_memory_bit_selected(2)

set_merged_nets

Merges the behavior of specified nets for delay analysis.

SYNTAX

```
status set_merged_nets
   net_list
```

Data Types

```
   net_list     list
```

ARGUMENTS

net_list

The list of nets to be merged.

DESCRIPTION

This command merges the behavior of specified nets. NanoTime eliminates all but one of the parallel arcs for timing analysis.

For example, for a high-fanout net such as a clock net, the clock tree might be implemented with multiple parallel drivers. To merge the behavior of multiple nets in the clock tree, list the net names in the **set_merged_nets** command, as in the following command:

nt_shell> **set_merged_nets {c1 c2 c3 c4}**

For delay analysis *starting* from any one of these nets, NanoTime eliminates all but one of the parallel timing arcs. For delay analysis *ending* at one of these nets, merging has no effect. Merging does not affect pattern matching, structure recognition, or the netlist.

To get a report on nets that have been merged, use the **report_merged_nets** command. To cancel the effects of the **set_merged_nets** command and restore independent behavior for the merged nets, use

the **remove_merged_nets** command.

COMMAND PHASING

The **set_merged_nets** command can only be executed between "netlist-linked" and "topology-checked" states. If used after "topology-checked", NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The first command merges the behavior of nets c1, c2, c3, and c4 for timing analysis. The second command reports the nets that share the same behavior as net c3 due to merging.

```
nt_shell> set_merged_nets {c1 c2 c3 c4}
nt_shell> report_merged_nets c3
```

SEE ALSO

remove_merged_nets(2)
report_merged_nets(2)

set_message_info

Set some information about diagnostic messages.

SYNTAX

```
string set_message_info -id message_id [-limit max_limit|-stop_on|-stop_ff]

string message_id
integer max_limit
```

ARGUMENTS

-id *message_id*

Information is to be set for the given *message_id*. The message must exist. Although different constraints allow different message types, no constraint allows severe or fatal messages.

-limit *max_limit*

Set the maximum number of occurrences for *message_id*. This is an integer greater than or equal to zero. If you set it to zero, that means the number of occurrences of the message is unlimited. Messages which occur after a limit is reached are automatically suppressed.

-stop_on

Force Tcl error if message is emitted.

-stop_off

Turn off a previous -stop_on directive

DESCRIPTION

The **set_message_info** command sets constraints on diagnostic messages (typically error, warning, and informational messages).

Currently, you can set a upper limit for the number of occurrences of a message. You can set this to zero to indicate that there is no limit. You can retrieve the current limit for a message using the **get_message_info** ***command. When the limit is exceeded, all future occurrences of the message are automatically suppressed. A count of total occurrences (including those suppressed) can be retrieved using get_message_info.***

EXAMPLES

The following example uses **set_message_info** to set a limit on the number of APP-027 messages to 100. When the 101st APP-027 message is about to be issued, you will be warned that the limit has been exceeded, and that all future occurrences will be suppressed.

```
prompt> set_message_info -id APP-027 -limit 100
prompt> do_command
Warning: can't find node U27.1 (APP-027)
Warning: can't find node U27.2 (APP-027)
Warning: can't find node U27.3 (APP-027)
 ...
Warning: can't find node U27.100 (APP-027)
Note - message 'APP-027' limit (100) exceeded.  Remainder will be suppressed.
1
```

SEE ALSO

```
get_message_info(2)
get_message_ids(2)
print_message_info(2)
suppress_message(2)
```

set_min_delay

Specifies a minimum delay for timing paths.

SYNTAX

```
status set_min_delay
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-use_existing_timing_points]
   delay_value
```

Data Types

```
   from_list           list
   rise_from_list      list
   fall_from_list      list
   through_list        list
   rise_through_list   list
   fall_through_list   list
   to_list             list
   rise_to_list        list
   fall_to_list        list
   delay_value         float
```

ARGUMENTS

-rise

Indicates that only rising path delays are to be constrained. If neither the **-rise** nor the **-fall** option is specified, both rising and falling delays are constrained.

-fall

Indicates that only falling path delays are to be constrained. If neither the **-rise** nor the **-fall** option is

specified, both rising and falling delays are constrained.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, or a pin. If a clock is specified, all registers and primary inputs related to that clock are used as path startpoints. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that the path must rise from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-fall_from ***fall_from_list***

Same as the **-from** option, except that the path must fall from the objects specified. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by falling edge of the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-from**, **-rise_from**, and **-fall_from** options.

-through ***through_list***

Specifies a list of pins, ports, and nets through which the paths must pass for maximum delay definition. Nets are interpreted to imply the leaf-level driver pins. If you omit the **-through** option, all timing paths specified using the **-from** and **-to** options are affected. You can specify the **-through** option more than one time in a single command invocation.

-rise_through ***rise_through_list***

Applies only to paths with a rising transition at the specified objects and is similar to the **-through** option. You can specify the *rise_through_list* option more than one time in a single command invocation.

-fall_through ***fall_through_list***

Applies only to paths and is similar to the **-through** option, but with a falling transition at the specified objects. You can specify the **-fall_through** option more than one time in a single command invocation.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a pin, or a net. Nets are interpreted to imply the leaf-level input pins. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, but applies only to paths rising at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edge of the clock at clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-fall_to ***fall_to_list***

Same as the **-to** option, but applies only to paths falling at the endpoint. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths launched by falling edge of

the clock at the clock source, taking into account any logical inversions along the clock path. You can use only one of the **-to**, **-rise_to**, and **-fall_to** options.

-use_existing_timing_points

When this option is used, the exception definition must only refer to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **set_max_delay** command has been issued.

delay_value

The required maximum delay value for the specified paths.

DESCRIPTION

This command specifies a required minimum delay for timing paths in the current design. The path length for any startpoint to any endpoint must be greater than the specified delay value. You must have a NanoTime Ultra license to use this feature.

Individual minimum delay targets are automatically derived from clock waveforms and port input or output delays. For more information, refer to the **create_clock**, **set_input_delay**, and **set_output_delay** command man pages.

The **set_min_delay** command is a point-to-point timing exception command. For example, the command overrides the default single-cycle timing relationship for one or more timing paths. Other point-to-point timing exception commands include the **set_multicycle_path**, **set_min_delay**, and **set_false_path** commands. A **set_max_delay** or **set_min_delay** command overrides a **set_multicycle_path** command.

The more general commands apply to more than one path. For example, if the **-from** option is used without a **-to** option, many paths would be selected.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. The following example specifies paths beginning at A1, passing through B1, then through C1, and ending at D1.

```
nt_shell> set_min_delay -from A1 -through B1 -through C1 -to D1
```

If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any of the objects. The following example specifies paths beginning at A1, passing through either B1 or B2, then passing through either C1 or C2, and ending at D1.

```
nt_shell> set_min_delay -from A1 -through {B1 B2} -through {C1 C2} -to D1
```

To list the **max_delay**, **min_delay**, **multicycle_path**, and **false_path** information for the design, use the **report_exceptions** command.

To remove information set by the **set_min_delay** command, use the **remove_min_delay** or **reset_design** command.

EXAMPLES

The following command specifies that any delay path to port Y must be greater than 10.0 units.

```
nt_shell> set_min_delay 10.0 -to {Y}
```

The following command specifies that all paths from ff1a or ff1b to ff2e must have delays greater than 15.0 units.

```
nt_shell> set_min_delay 15.0 -from {ff1a ff1b} -to {ff2e}
```

The following example specifies that all paths to endpoints clocked by PHI2 must have delays greater than 8.5 units.

```
nt_shell> set_min_delay 8.5 -to [get_clocks PHI2]
```

The following example sets a requirement that all paths leading to ports named busA[*] must have delays greater than 5.0.

```
nt_shell> set_min_delay 5.0 -to "busA[*]"
```

The following example specifies that all timing paths from ff1/CP to ff2/D that pass through one or more of {U1/Z U2/Z} and one or more of {U3/Z U4/C} must have delays greater than 8.0 units.

```
nt_shell> set_min_delay 8.0 -from ff1/CP \
          -through {U1/Z U2/Z} -through {U3/Z U4/C} -to ff2/D
```

The following example specifies that all timing paths from ff1/CP to ff2/D that rise through one or more of {U1/Z U2/Z} and fall through one or more of {U3/Z U4/C} must have delays greater than 8.0 units.

```
nt_shell> set_min_delay 8.0 -from ff1/CP \
          -rise_through {U1/Z U2/Z}\
          -fall_through {U3/Z U4/C} -to ff2/D
```

SEE ALSO

create_clock(2)
remove_min_delay(2)
set_max_delay(2)
remove_max_delay(2)
reset_design(2)
set_false_path(2)
set_input_delay(2)
set_multicycle_path(2)
set_output_delay(2)

set_min_library

Sets the .db libraries to be used for minimum-delay and maximum-delay analysis.

SYNTAX

```
status set_min_library
   [-min_version min_library | -none]
   max_library
```

Data Types

```
   min_library          string
   max_library          string
```

ARGUMENTS

-min_version *min_library*

Specifies the .db library to be used for minimum-delay analysis.

-none

Causes NanoTime to use the specified library for both minimum-delay and maximum-delay analysis

max_library

The .db library to be used for maximum-delay analysis.

DESCRIPTION

This command creates a max/min relationship between two .db libraries. Once the relationship is established, the max_library is used for maximum-delay analysis and the min_library is used for minimum-delay analysis of cells contained in the two libraries.

For each library cell in the *max_library*, the **set_min_library** command searches for a library cell of the same name in the *min_library*. If it is not found, a warning is issued, and there is no max/min relationship

for that library cell.

If the *min_library* does have the named library cell, the command compares the two library cells to verify that they have the same pins (in the same order, with the same direction) and the same timing arcs. If any of these conditions fails, a warning is issued, and there is no max/min relationship for that library cell.

At least one library cell must match for the command to succeed.

When NanoTime needs to compute a minimum delay value, if there is a max/min relationship for the library cell, the timing information from the *min_library* is used. Otherwise, the information is taken from the *max_library*.

Only the *max_library* specified in the **set_min_library** command should be in the link path. The library used as the *min_library* must not be placed in the link path.

The **list_libs** command shows a max/min relationship with the letter "M" to indicate the max library and lowercase "m" to indicate the minimum library. In addition, the **report_lib** command shows any min library associated with a max library.

COMMAND PHASING

The **set_min_library** command can only be executed between the "netlist-linked" and "design-checked" states. If used after "design-checked", NanoTime transitions back to the "design-checked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The following is a typical session using the **set_min_library** command.

```
nt_shell> set search_path ". /data"
. /data
nt_shell> set link_path "* typical"
* typical
nt_shell> read_library CL180G/libcell/typical.db
Loading db file '/data/CL180G/libcell/typical.db'
1
nt_shell> read_library CL180G/libcell/slow.db
Loading db file '/data/CL180G/libcell/slow.db'
1
nt_shell> read_library CL180G/libcell/fast.db
Loading db file '/data/CL180G/libcell/fast.db'
1
nt_shell> register_netlist -format verilog rpt_cell.v
1
nt_shell> link_design rpt_cell
Compiling "rpt_cell.v"
Linking design rpt_cell...
Design 'rpt_cell' was successfully linked.
1
nt_shell> set_min_library -min_version fast typical
Created max/min library relationship:
  Max: /data/CL180G/libcell/typical.db:typical
  Min: /data/CL180G/libcell/fast.db:fast
1
```

```
nt_shell> list_libraries
Library Registry:
    builtin_elements   builtin_elements:None
  m fast     /data/CL180G/libcell/fast.db:fast
    slow     /data/CL180G/libcell/slow.db:slow
  M typical  /data/CL180G/libcell/typical.db:typical

1
nt_shell> report_lib typical
 ...
Operating Conditions:

  Name              Process      Temp   Voltage
  --------------- --------- --------- ---------
  typical            1.0000   25.0000    1.8000

Min Library:

  /data/CL180G/libcell/fast.db:fast
 ...
Lib Cell        Attributes
--------------- ----------
OAI211X0        --
AOI2BB1X0       --
AFHCONX1        --
 ...
```

SEE ALSO

list_libs(2)
report_design(2)
report_lib(2)
set_technology(2)
link_path(3)

set_min_pulse_width

Sets a minimum pulse width constraint for specified objects in the design.

SYNTAX

```
status set_min_pulse_width
   -low | -high
   [-use_existing_timing_points]
   check_value
   object_list
```

Data Types

```
   check_value       float
   object_list       list
```

ARGUMENTS

-low

Specifies the minimum pulse width constraint for low pulses (from high to low to high). Either the **-low** or **-high** option must be specified.

-high

Specifies the minimum pulse width constraint for high pulses (from low to high to low). Either the **-low** or **-high** option must be specified.

-use_existing_timing_points

When this option is used the pins in the *object_list* must already be timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute has been set to **true** for the listed pins. When this option is used then the **check_design** command does not need to be run again after the **set_min_pulse_width** command has been issued.

check_value

A positive floating-point value that specifies the minimum pulse width, in time units.

object_list

Specifies the list of clocks, cells, or pins in the current design to which the constraint applies.

DESCRIPTION

The **set_min_pulse_width** command specifies the minimum allowable pulse width for all signals in a clock tree, for all pins of specified cells, or for specified pins. This constraint must be set before the **check_design** command is executed. A pulse width less than the specified value triggers a pulse width violation. This condition is detected during path tracing.

Minimum pulse width constraints on pins are evaluated using reference and checked timing paths from the same start pin, if such paths available, or the same start net. If paths from the same start pin or net are not available then the minimum pulse width constraints on pins will not be evaluated and appropriate warning messages will be issued.

A clock pulse width violation can occur if the negation of the clock signal happens too soon after the assertion of the clock signal. Clock pulse width violations can prevent sequential devices from capturing data properly.

If you set a minimum pulse width constraint on a cell and the cell also has a minimum pulse width constraint defined in the technology library, NanoTime uses the more restrictive (smaller) value.

To generate a report on pulse width constraints, use the **report_min_pulse_width** or **report_constraint -min_pulse_width** command. This can be done after executing the **trace_paths** command.

To remove minimum pulse width constraints that have been set, use the **remove_min_pulse_width** command.

COMMAND PHASING

The **set_min_pulse_width** command can only be executed between "netlist-linked" and "topology-checked" states. If used after "topology-checked", NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed in the later phase(s).

However, if you use the **-use_existing_timing_points** option, you can use the **set_min_pulse_width** command after the **check_design** command has been executed.

EXAMPLES

The following example sets a minimum pulse width requirement of 2.0 for both low and high pulses of clock signal CK1.

```
nt_shell> set_min_pulse_width 2.0 [get_clocks CK1]
```

The following example sets a minimum pulse width requirement of 2.5 for the low pulses of the clock

signal on pin U1/Z.

```
nt_shell> set_min_pulse_width -low 2.5 U1/Z
```

SEE ALSO

remove_min_pulse_width(2)
report_constraint(2)
report_min_pulse_width(2)

set_model_input_transition_indexes

Sets the transition time index values for input ports and bidirectional ports used by the **extract_model** command to generate the model lookup tables.

SYNTAX

```
status set_model_input_transition_indexes
   [-min]
   [-max]
   [-rise]
   [-fall]
   [-nominal time]
   [-rail_voltage volts]
   index_list
   port_list
```

Data Types

```
   time            float
   volts           float
   index_list      list
   port_list       list
```

ARGUMENTS

-min

Specifies that the index values are to be used for modeling the minimum (shortest) transitions.

-max

Specifies that the index values are to be used for modeling the maximum (longest) transitions.

-rise

Specifies that the index values are to be used only for rising transitions.

-fall

Specifies that the index values are to be used only for falling transitions.

-nominal *time*

Sets the nominal transition time used to find the critical paths. The nominal value does not have to be in the *index_list* argument. The default is the median value of the given set of index values.

-rail_voltage *volts*

Sets the rail voltage for the input transition used to find the critical paths. The default is the value of the **oc_global_voltage** variable.

index_list

A list of index values (floating-point numbers greater than or equal to 0.0) in time units. This option works with the **-min**, **-max**, **-rise**, and **-fall** options to specify transition index values for specific model tables. The list must contain at least three values.

port_list

The list of input ports and bidirectional ports affected by the command, including clock ports.

DESCRIPTION

The **set_model_input_transition_indexes** command sets the input transition values to be used for model extraction with the **extract_model** command. The list of values determines the range and spacing of transition times in the model lookup tables for the specified inputs.

The nominal value (which can be set with the **-nominal** option) is used in the initial collection of paths for the model. However, it does not appear as a table index in the model unless it is explicitly included in the *index_list* argument.

In the absence of the **set_model_input_transition_indexes** command, NanoTime uses the index values specified by the **model_default_input_transition_indexes** variable.

In the extracted model, the clock inputs use only the nominal value from the input transition index values, rather than the full range of values. This behavior is based on the assumption that the signal on a clock port is much more consistent and predictable than for data ports.

To cancel the effects of the **set_model_input_transition_indexes** command, set the indexes to the same values as those in the **model_default_input_transition_indexes** variable, or use the **reset_design** command.

COMMAND PHASING

The **set_model_input_transition_indexes** command can be used any time after the **link_design** and before the **extract_model** or **trace_paths** command. If you use the **set_model_input_transition_indexes** command after the **extract_model** or **trace_paths** command, NanoTime transitions back to the "design-checked" state and discards all commands and operations performed after the **extract_model** or **trace_paths** command.

EXAMPLES

The following command sets the input transition values to 0.0, 0.5, and 1.0 in the minimum-slew model lookup tables and sets the nominal value to 0.8, for all inputs in the design.

```
nt_shell> set_model_input_transition_indexes \
          -max -nominal 0.8 {0.1 0.5 1.0} \
          [all_inputs]
```

SEE ALSO

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

set_model_load_indexes

Sets the load index values at output ports and bidirectional ports, used by the **extract_model** command to generate the model lookup tables.

SYNTAX

```
status set_model_load_indexes
   [-min]
   [-max]
   index_list
   port_list
```

Data Types

```
   index_list          list
   port_list           list
```

ARGUMENTS

-min

The index values for modeling the minimum capacitance.

-max

The index values for modeling the maximum capacitance.

index_list

List of index values (floating-point numbers greater than or equal to 0.0) in capacitance units. Works with the **-min** and **-max** options to specify load index values for specific model tables. The list must contain at least three values.

port_list

The list of output ports and bidirectional ports affected by the command.

DESCRIPTION

The **set_model_load_indexes** command sets the load values to be used for model extraction with the **extract_model** command. The list of values determines the range and spacing of load values in the model lookup tables for the specified outputs.

In the absence of the **set_model_load_indexes**command, NanoTime uses the index values specified by the **model_default_load_indexes** variable.

In the **extract_model** command, if the **-library_elements** option is set to **{nldm ccs_timing}**, and the design for which you are generating an extracted model contains library cells, and a CCS-based library cell drives an output port of the design, NanoTime uses the load index values of the existing CCS driver model. Under these conditions, NanoTime ignores the load index values specified by the **set_model_load_indexes** command and the **model_default_load_indexes** variable.

During path tracing for model generation, NanoTime uses nominal load values set with the **set_load** command.

To cancel the effects of the **set_model_load_indexes** command, set the indexes to the same values as those in the **model_default_load_indexes** variable, or use the **reset_design** command.

COMMAND PHASING

The **set_model_load_indexes** command can be used any time after the **link_design** and before the **extract_model** or **trace_paths** command. If you use the **set_model_load_indexes** command after the **extract_model** or **trace_paths** command, NanoTime transitions back to the "design-checked" state and discards all commands and operations performed after the **extract_model** or **trace_paths** command.

EXAMPLES

The following command sets the output load values to 0.2, 0.4, and 0.6 in the maximum-capacitance model lookup tables for all outputs in the design.

```
nt_shell> set_model_load_indexes -max \
            {0.2 0.4 0.6} [all_outputs]
```

SEE ALSO

```
set_load(2)
set_model_input_transition_indexes(2)
```

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

set_multicycle_path

Identifies multicycle paths in the design.

SYNTAX

```
status set_multicycle_path
   [-setup]
   [-hold]
   [-hold_cycle]
   [-rise]
   [-fall]
   [-start]
   [-end]
   [-from from_list
     | -rise_from rise_from_list
     | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-consecutive]
   [-use_existing_timing_points]
   path_multiplier
```

Data Types

from_list	list
rise_from_list	list
fall_from_list	list
through_list	list
rise_through_list	list
fall_through_list	list
to_list	list
rise_to_list	list
fall_to_list	list
path_multiplier	integer

ARGUMENTS

-setup

Specifies the number of clock cycles for setup (maximum delay) timing checks. Hold calculations are not affected by the command.

-hold

Specifies the number of clock cycles for hold (minimum delay) timing checks. Setup calculations are not affected by the command. For a larger *path_multiplier* value, the hold check is shifted to an earlier time relative to the actual setup check.

-hold_cycle

Specifies the number of clock cycles for hold (minimum delay) timing checks. Setup calculations are not affected by the command. For a larger *path_multiplier* value, the hold check is shifted to a later time relative to the default hold check. If the **-setup**, **-hold**, and **-hold_cycle** options are all omitted from the command, the *path_multiplier* argument applies to both the **-setup** and **-hold_cycle** options.

-rise

Specifies that only the paths with a rising value at the path endpoint are affected.

-fall

Specifies that only the paths with a falling value at the path endpoint are affected. If neither the **-rise** nor the **-fall** option is specified, timing paths with both rising and falling values are affected.

-start

Specifies that the amount of the multicycle timing shift is based on the period of the start clock (the clock that launches data at the path startpoint).

-end

Specifies that the amount of the multicycle timing shift is based on the period of the end clock (the clock that captures data at the path endpoint). The **-start** or **-end** option has an effect only if the startpoint and endpoint clocks have different periods. The default behavior is to move a setup check relative to the end clock, or a hold check relative to the start clock.

-from *from_list*

Specifies a list of path startpoint objects. A valid timing startpoint is a clock, a primary input or bidirectional port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has an input delay specified. If you specify a clock, the command affects all paths that start at sequential elements and primary inputs clocked by that clock. If you specify a cell, the command affects all paths that start at input pins and bidirectional pins of the cell. You can use at most one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from *rise_from_list*

Same as the **-from** option, except that only paths with rising-edge signals from the specified objects are affected. If you specify a clock, the command affects all paths that start at sequential elements and primary inputs clocked by the rising edge of that clock at the source (which could result in a rising or falling edge at the path startpoint, depending on logical inversions along the clock path.)

-fall_from *fall_from_list*

Same as the **-rise_from** option, except that only paths with falling-edge signals from the specified objects are affected.

-through *through_list*

Specifies a list of pins, ports, cells, or nets through which the paths must pass to be affected by the command. You can specify the **-through** option more than one time in the same command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net.

-rise_through *rise_through_list*

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through *fall_through_list*

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to *to_list*

Specifies a list of path endpoint objects. A valid timing endpoint is a clock, a primary output or bidirectional port, a sequential cell, a data pin of a sequential cell, or a pin that has an output delay specified. If you specify a clock, the command affects all paths that end at sequential elements and primary outputs clocked by that clock. If you specify a cell, the command affects all paths that end at input pins and bidirectional pins of the cell. You can use at most one of the **-to**, **-rise_to**, and **-fall_to** options.

-rise_to *rise_to_list*

Same as the **-to** option, except that only paths with rising-edge signals to the specified objects are affected. If you specify a clock, the command affects all paths that end at sequential elements and primary outputs clocked by the rising edge of that clock at the source (which could result in a rising or falling edge at the path endpoint, depending on logical inversions along the clock path.)

-fall_to *fall_to_list*

Same as the **-rise_to** option, except that only paths with falling-edge signals arriving at the specified objects are affected.

path_multiplier

An integer that specifies the number of clock cycles to move the setup or hold timing check relative to the default timing position. For the **-setup** option, the setup check is performed when the specified number of cycles of the capture (end) clock occur after the launch of data from the path startpoint. The default multiplier value for the **-setup** option is 1.

Using the **-hold** option adjusts the position of the hold check according to the position of the setup check; the hold check is shifted earlier by the specified number of clock cycles from the position at one cycle before the actual setup check. For the **-hold_cycle** option, the hold check is performed later by the specified number of cycles of the launch (start) clock; the hold check is shifted to the right from the default hold check position. The default multiplier value for the **-hold_cycle** option is 1.

-consecutive

Specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

-use_existing_timing_points

When this option is used the exception definition must only refer to pins which are already timing points.

This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **set_multicycle_path** command has been issued.

DESCRIPTION

The **set_multicycle_path** command specifies the number of clock cycles required between the launch and capture of data in specified timing paths. Use this command to modify the times at which setup and hold checks are performed.

This command defines a path-based timing exception, along with the **set_no_same_phase_path**, **set_same_phase_path**, **set_false_path**, **set_no_timing_check_path**, and **set_exclude_reference_path** commands. Path-based timing exceptions have priority over object-based timing exceptions, such as those set with the **set_phasing** or **set_timing_check_attributes** commands or invoked via the **timing_default_phasing** variable.

In case of conflicting exceptions for a path, the timing exception types have the following order of priority, from highest to lowest:

```
1. set_false_path
2. set_no_timing_check_path
3. set_multicycle_path
4. set_no_same_phase_path
5. set_same_phase_path
```

Multiple exception settings that are not in conflict with each other can apply to a single path. There is no conflict between the **set_multicycle_path** command and the **set_same_phase_path** or **set_no_same_phase_path** commands. Also, there is no conflict between the **-setup** and **-hold** settings for a specific exception.

To undo the effects of a **set_multicycle_path** command, use the **remove_multicycle_path** command.

NanoTime applies certain rules to determine timing relationships for paths between clocked elements. The rules are based on active edges of the clocks. For latches, the opening edge launches data and the closing edge latches data. For flip-flops, a single active edge both launches and captures data.

The clock edges that are considered the launch and capture clock edges depend on several conditions, including the relative timing of the launch and capture clocks, the types of sequential elements (transparent latches or flip-flops), the same-cycle or next-cycle settings for the sequential elements or the path, and the **timing_intersection_transparency** variable.

For designs with multiple clocks, the launch and capture clocks can be different, resulting in multiple setup and hold relationships. The nearest capture edge that follows a launch edge determines the setup requirement for the path. The most restrictive hold relationship between a launch edge and a capture edge determines the hold requirement for the path.

The setup and hold timing relationships can be changed by using the **set_multicycle_path** command and by choosing same-cycle or next-cycle checking. If both methods are used, the effects are added together. For example, if you invoke next-cycle checking to move the setup check forward by one cycle and also use the **set_multicycle_path** command to move the setup check forward by one cycle, the total effect is to move the check forward two cycles.

Setup, Hold and Path Multiplier

By default, NanoTime performs each setup (maximum delay) check at the first capture edge that occurs after the launch edge. To modify this check, use the **-setup** option and specify the number of capture edges after launch were the setup check is to be done. The number you specify is called the path multiplier.

For example, the **-setup 1** option has no effect because it causes the setup check to occur at the first capture edge after launch, the same as the default. Use the **-setup 2** option to perform the check at the second capture edge, in other words, to move the setup check one clock cycle later than the default. Use the **-setup 3** option to move the setup check two clock cycles later than the default, and so on. The larger the number, the later the setup check is performed and the easier it is for the setup constraint to be satisfied.

Moving the setup check with the **-setup** option does not affect the timing of the hold (minimum delay) check. To modify a hold check, use either the **-hold** or **-hold_cycle** option.

The **-hold** option moves the hold timing check backward in time from the position at one clock cycle before the current setup check. For example, the **-hold 0** option causes the hold check to occur one clock cycle before the setup check, including a setup check that has been moved with the **-setup** option of another **set_multicycle_path** command. Using the **-hold 1** option causes the hold check to occur two clock cycles before the setup check, and so on. The larger the number, the earlier the hold check is performed and the easier it is for the hold constraint to be satisfied.

The **-hold_cycle** option moves the hold timing check forward in time from its default position, irrespective of any movement of the setup check. Using the **-hold_cycle 0** option causes the hold check to occur one clock cycle earlier than the default time. Using the **-hold_cycle 1** option causes the hold check to occur at the default time, and therefore has no effect. Using the **-hold_cycle 2** option causes the hold check to occur one clock cycle later than the default time, and so on. The larger the number, the later the hold check is performed and the harder it is for the hold timing constraint to be satisfied.

When you use the **-hold** option, NanoTime sets the hold check time relative to the where the setup check is being performed. However, when you run the **trace_paths** command, NanoTime internally converts the **-hold** setting to the equivalent **-hold_cycle** setting. (You can see this change if you use the **report_exceptions** command.) As a result, if you change the setup time after you run the **trace_paths** command, the hold check stays the same and is not based on the setup time.

If the launch (start) clock and capture (end) clock are two different clocks with different frequencies, you should consider which clock is being used to move a setup or hold check. By default, NanoTime moves a setup check by a multiplier of the end clock period, but it moves a hold check by a multiplier of the start clock period. If this behavior is not what you want, use the **-start** or **-end** option to explicitly specify which clock period to use for adjusting the setup or hold check.

From, Through, and To Points

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any one of the listed objects. The **-consecutive** option specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

To restrict the multicycle path timing exception to paths with only rising edges or only falling edges at the path endpoint, use the **-rise** or **-fall** option. These options can be used even when there is no "to"

type option specified.

NOTE: The **set_multicycle_path** command is a Synopsys Design Constraints (SDC) command. However, the **-rise_from**, **-rise_to**, **-rise_through**, **-fall_from**, **-fall_to**, **fall_through**, **-consecutive**, and **-hold_cycle** options might not be supported by other tools that use SDC commands. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_multicycle_path** command is typically used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option with the **set_multicycle_path** command, you can execute it after the **check_design** command and before the **trace_paths** command or **extract_model** command.

EXAMPLES

The following example sets all paths between latch1b and latch2d to two-cycle paths for setup, which is one cycle later than the default. Hold checks are not affected.

```
nt_shell> set_multicycle_path 2 \
          -from {latch1b} -to {latch2d}
```

The following example uses the **-start** option to specify a three-cycle setup path relative to the clock at the path startpoint. Clock sources are specified, affecting all sequential elements clocked by those clocks, and also ports with input delay or output delay relative to those clocks.

```
nt_shell> set_multicycle_path -setup 3 -start \
          -from {clk50mhz} -to {clk10mhz}
```

The following commands set all rising paths ending at ff12/D to two cycles, and falling paths ending at ff12/D to one cycle.

```
nt_shell> set_multicycle_path 2 -rise_to {ff12/D}
nt_shell> set_multicycle_path 1 -fall_to {ff12/D}
```

The following multifrequency example shows a 20-ns clock to 10-ns clock with a multicycle path of 2. Assume a path from ff1 (clocked by CLK2) to ff2 (clocked by CLK1).

```
nt_shell> create_clock -period 20 \
          -waveform {0 10} CLK2
nt_shell> create_clock -period 10 \
          -waveform {0 5} CLK1
nt_shell> set_multicycle_path 2 -setup \
          -from ff1/CP -to ff2/D
```

The single-cycle setup relation is from the CLK1 edge at 0 ns to the CLK2 edge at 10 ns. With the multicycle path, the setup relation is now 20 ns (from CLK1 edge at 0 ns to CLK2 edge at 20 ns). The hold check is not affected.

The following example sets all timing paths starting from ff1/CP and ending at ff2/D that pass through one or more of points U1/Z and U2/Z and one or more of points U3/Z and U4/C to two-cycle paths for setup.

```
nt_shell> set_multicycle_path 2 \
          -from ff1/CP \
          -through {U1/Z U2/Z} \
          -through {U3/Z U4/C} \
          -to ff2/D
```

SEE ALSO

remove_multicycle_path(2)
report_exceptions(2)
reset_design(2)
set_exclude_reference_path(2)
set_false_path(2)
set_no_same_phase_path(2)
set_no_timing_check_path(2)
set_same_phase_path(2)
set_phasing(2)
timing_default_phasing(3)
timing_intersection_transparency(3)

set_no_same_phase_path

Sets no-same-phase checking on specified paths, causing NanoTime to check for data arrival in the next clock phase rather than the same clock phase.

SYNTAX

```
status set_no_same_phase_path
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-consecutive]
   [-use_existing_timing_points]
```

Data Types

```
   from_list           list
   rise_from_list      list
   fall_from_list      list
   through_list        list
   rise_through_list   list
   fall_through_list   list
   to_list             list
   rise_to_list        list
   fall_to_list        list
```

ARGUMENTS

-setup

Sets no-same-phase checking for setup checks only. In the absence of the **-setup** and **-hold** options, no-same-phase checking applies to both setup and hold checks.

-hold

Sets no-same-phase checking for hold checks only.

-rise

Specifies that only paths with a rising transition at the path endpoint are affected. In the absence of any of the **-rise**, **-fall**, or related options, both rising and falling edges are affected.

-fall

Specifies that only paths with a falling transition at the path endpoint are affected.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all sequential elements and primary inputs related to that clock are used as path startpoints. If you specify a cell, all input pins and bidirectional pins of the cell are used as path startpoints. You can use at most one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only rising-edge signals from the specified objects are affected. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from ***fall_from_list***

Same as the **-rise_from** option, except that only falling-edge signals from the specified objects are affected.

-through ***through_list***

Specifies a list of pins, ports, cells, or nets through which the paths must pass. You can use the **-through** option more than one time in one command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net.

-rise_through ***rise_through_list***

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through ***fall_through_list***

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, all input pins and bidirectional pins of the cell are used as path endpoints. You can use at most one of the **-to**, **-rise_to**, and **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge signals to the specified objects are affected. If a

clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edges of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to ***fall_to_list***

Same as the **-rise_to** option, except that only falling-edge signals to the specified objects are affected.

-consecutive

Specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

-use_existing_timing_points

When this option is used the path definition must only refer to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, then the **check_design** command does not need to be run again after the **set_no_same_phase_path** command has been issued.

DESCRIPTION

The **set_no_same_phase_path** command specifies no-same-phase checking for all timing paths that meet the specified the criteria in the **-from**, **-through**, **-to**, and related options.

This command defines a path-based timing exception, along with the **set_same_phase_path**, **set_multicycle_path**, **set_false_path**, **set_no_timing_check_path**, and **set_exclude_reference_path** commands. Path-based timing exceptions have priority over object-based timing exceptions, such as those set with the **set_phasing** or **set_timing_check_attributes** commands or invoked via the **timing_default_phasing** \variable.

In case of conflicting exceptions for a path, the timing exception types have the following order of priority, from highest to lowest:

```
1. set_false_path
2. set_no_timing_check_path
3. set_multicycle_path
4. set_no_same_phase_path
5. set_same_phase_path
```

Multiple exception settings that are not in conflict with each other can apply to a single path. There is no conflict between the **set_multicycle_path** command and the **set_same_phase_path** or **set_no_same_phase_path** commands. Also, there is no conflict between the **-setup** and **-hold** settings for a specific exception.

To undo the effects of a **set_no_same_phase_path** command, use the **remove_no_same_phase_path** command.

Same-Phase and No-Same-Phase 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-phase 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 following clock cycle rather than the same clock cycle. (This behavior can be changed by setting the **timing_intersection_transparency** variable.)

Under same-phase 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 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.

Under same-phase checking conditions, if a downstream latch is designed to capture data in the next clock phase rather than the same clock phase, you need to set no-same-phase checking so that the timing checks are performed at the correct times. In 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.

How to Specify Phasing

There are several ways to specify same-phase or no-same-phase checking, listed here in order from highest to lowest priority:

- Use the **set_same_phase_path** and **set_no_same_phase_path** commands to define the behavior for specific objects.
- Use the **set_phasing** command to specify the behavior for instances of latches, precharge gates, and output ports.
- Use the **set_timing_check_attributes** command with the **-force_same_phase** or **-force_no_same_phase** option to define the behavior for specific timing checks.
- Use the **timing_default_phasing** variable, which specifies the behavior of all latches, precharge gates, and output ports when those objects are created or defined.

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** type options in the command specify the paths that are to use same-phase or no-same-phase checking, irrespective of the settings on objects in the paths.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any one of the listed objects. The **-consecutive** option specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

COMMAND PHASING

The **set_no_same_phase_path** command is typically used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option with the **set_no_same_phase_path** command, you can execute it after the **check_design** command and before the **trace_paths** command or **extract_model** command.

EXAMPLES

The following command sets no-same-phase checking for all timing paths from port in1 to port outd3.

```
nt_shell> set_no_same_phase_path \
          -from [get_ports in1] \
          -to [get_ports outd3]
```

SEE ALSO

remove_no_same_phase_path(2)
report_exceptions(2)
set_exclude_reference_path(2)
set_false_path(2)
set_multicycle_path(2)
set_no_timing_check_path(2)
set_same_phase_path(2)
timing_intersection_transparency(3)
timing_default_phasing(3)
set_phasing(2)
remove_phasing(2)

set_no_timing_check_path

Disables timing checks at specified points in the design while allowing path tracing to proceed through those points.

SYNTAX

```
status set_no_timing_check_path
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-consecutive]
   [-use_existing_timing_points]
```

Data Types

```
   from_list            list
   rise_from_list       list
   fall_from_list       list
   through_list         list
   rise_through_list    list
   fall_through_list    list
   to_list              list
   rise_to_list         list
   fall_to_list         list
```

ARGUMENTS

-setup

Specifies that timing checks of any type (setup or hold checks) should not be performed for maximum delay analysis. If you do not specify either the **-setup** or **-hold** option, timing checks are disabled for both minimum and maximum delay analysis.

-hold

Specifies that timing checks should not be performed for minimum delay analysis.

-rise

Specifies that paths with rising delays at the path endpoint are affected. If you do not specify either of the **-rise** or **-fall** options, both rise and fall timing are affected.

-fall

Specifies that paths with falling delays at the path endpoint are affected.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all sequential elements and primary inputs related to that clock are used as path startpoints. If you specify a cell, all input pins and bidirectional pins of that cell are affected. You can use at most one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only rising-edge signals from the specified objects are affected. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from ***fall_from_list***

Same as the **-rise_from** option, except that only falling-edge signals from the specified objects are affected.

-through ***through_list***

Specifies a list of pins, ports, cells, or nets through which the paths must pass. You can specify **-through** more than one time in one command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net. If you do not specify any type of **through** option, all timing paths specified using the **-from** and **-to** options are affected.

-rise_through ***rise_through_list***

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through ***fall_through_list***

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all sequential elements and primary outputs related to that clock are used as path endpoints. If you specify a cell, all input pins and bidirectional pins of that cell are affected. You can use at most one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge signals to the specified objects are affected. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edges of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to ***fall_to_list***

Same as the **-rise_to** option, except that only falling-edge signals to the specified objects are affected.

-consecutive

Specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

-use_existing_timing_points

When this option is used the exception definition must refer only to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, the **check_design** command does not need to be run again after the **set_no_timing_check_path** command has been issued.

DESCRIPTION

The **set_no_timing_check_path** command disables timing checks at specified locations in the design.

This command defines a path-based timing exception, along with the **set_no_same_phase_path**, **set_same_phase_path**, **set_false_path**, **set_multicycle_path**, and **set_exclude_reference_path** commands. Path-based timing exceptions have priority over object-based timing exceptions, such as those set with the **set_phasing** or **set_timing_check_attributes** commands or invoked via the **timing_default_phasing** variable.

In case of conflicting exceptions for a path, the timing exception types have the following order of priority, from highest to lowest:

```
1. set_false_path
2. set_no_timing_check_path
3. set_multicycle_path
4. set_no_same_phase_path
5. set_same_phase_path
```

Multiple exception settings that are not in conflict with each other can apply to a single path. There is no conflict between the **set_multicycle_path** command and the **set_same_phase_path** or **set_no_same_phase_path** commands. Also, there is no conflict between the **-setup** and **-hold** settings for a specific exception.

To undo the effects of a **set_no_timing_check_path** command, use the **remove_no_timing_check_path** command.

The **set_no_timing_check_path** command must specify at least one path endpoint using the **-to**, **-rise_to**, or **-fall_to** options. No timing checks are performed at the specified endpoints. Instead, they are treated like plain combinational logic. If a specified endpoint is a latch, paths that end at that latch are not

saved into the path database. Path tracing continues through the latch as if it were transparent, but without any cycle accounting adjustments.

In other words, NanoTime first finds paths that match the **-to**, **-rise_to**, or **-fall_to** options. NanoTime evaluates all timing checks up to the point where the options are matched. Timing checks for the next latch (or other topology that includes timing checks) in the path are not evaluated. Note that more than one timing check might be eliminated, depending on the topology structure. Path tracing continues beyond the affected topology and all timing checks in the rest of the path are evaluated.

This command is different from the **set_false_path** command, which causes the specified paths to be ignored completely and stops all path tracing through them. The **set_no_timing_check_path** command does not stop path tracing, but merely prevents timing checks from being performed at the specified endpoints.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any one of the listed objects. The **-consecutive** option specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

COMMAND PHASING

The **set_no_timing_check_path** command should be used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option with the **set_no_timing_check_path** command, you can execute it after the **check_design** command and before the **trace_paths** command or **extract_model** command.

EXAMPLES

A domino precharge circuit uses two different evaluate clocks, ck1 and ck2. By default, NanoTime performs a timing check between data arriving at the precharge node launched by one clock and captured by the other, but this situation cannot actually occur in the circuit. To prevent the check from being done, use the following commands:

```
nt_shell> set_no_timing_check_path \
          -from [get_clocks ck1] \
          -to [get_clocks ck2]
nt_shell> set_no_timing_check_path \
          -from [get_clocks ck2] \
          -to [get_clocks ck1]
```

The following command disables timing checks from pin U14.Z to pin ff29.Reset for signals with rising transitions at the startpoint:

```
nt_shell> set_no_timing_check_path \
          -rise_from [get_pins U14.Z] \
          -to [get_pins ff29.Reset]
```

The following command disables timing checks for minimum delay analysis for paths with endpoints clocked by CLK1.

```
nt_shell> set_no_timing_check_path -hold \
          -to [get_clocks CLK1]
```

The following command disables timing checks for paths from ff1.CP to ff2.D that pass through one or more of {U1.Z U2.Z} and one or more of {U3.Z U4.C}.

```
nt_shell> set_no_timing_check_path -from ff1.CP \
          -through {U1.Z U2.Z} \
          -through {U3.Z U4.C} \
          -to ff2.D
```

SEE ALSO

remove_no_timing_check_path(2)
report_exceptions(2)
set_exclude_reference_path(2)
set_false_path(2)
set_multicycle_path(2)
set_no_same_phase_path(2)
set_same_phase_path(2)

set_noise_margin

Sets a new noise margin on a list of objects (nets, ports, or pins).

SYNTAX

```
status set_noise_margin
   [-above]
   [-below]
   [-high]
   [-low]
   value
   objects
```

Data Types

```
   value      float
   objects    list
```

ARGUMENTS

-above

Specifies the noise margin for values going above the stable signal level (either high or low) of the victim net.

-below

Specifies the noise margin for values going below the stable signal level (either high or low) of the victim net.

-high

Specifies the noise margin for values related to the high signal level of the victim net.

-low

Specifies the noise margin for values related to the low signal level of the victim net.

value

The new noise margin.

objects

Specifies the objects in the design to which a new noise margin applies. Nets map to the set of leaf pins connected to that net.

DESCRIPTION

The **set_noise_margin** command specifies a noise margin on a selected set of pins. This command can be used to replace one or more of the global noise margin values on a specific set of pins in the design. The **-above**, **-below**, **-high**, and **-low** options can be used in combination to specify which noise margins to set.

In the absence of an object-specific noise margin, NanoTime uses values from these four global variables to compute noise slack values on a pin:

si_noise_margin_above_high
si_noise_margin_above_low
si_noise_margin_below_high
si_noise_margin_below_low

COMMAND PHASING

The **set_noise_margin** command can be executed any time after the **check_design** command.

EXAMPLES

The following command sets a noise margin of 0.1 volts on pin Xxor7.MP1.g .

```
nt_shell> set_noise_margin 0.1 Xxor7.MP1.g
```

The following command sets a noise margin of 0.1 volts for noise above the ground rail on pin Xxor2.MN1.g .

```
nt_shell> set_noise_margin -above -low 0.1 Xxor2.MN1.g
```

SEE ALSO

report_noise(2)
si_enable_noise_analysis(3)
si_noise_margin_above_high(3)
si_noise_margin_above_low(3)

```
si_noise_margin_below_high(3)
si_noise_margin_below_low(3)
```

set_noise_parameters

Sets parameters that influence how signal integrity (crosstalk) noise values are computed and the types of noise computed.

SYNTAX

```
set_noise_parameters
   [-include_beyond_rails]
```

ARGUMENTS

-include_beyond_rails

Directs NanoTime to compute noise values beyond the rails. This causes the noise types above_high and below_low to be computed.

DESCRIPTION

The **set_noise_parameters** command with the **-include_beyond_rails** option forces analysis of signal integrity (crosstalk) noise to include noise values outside the power rails. By default, only values within the rails are computed, namely the above_low and below_high noise types.

COMMAND PHASING

The **set_noise_parameters** command can be executed any time after the **check_design** command but before the **update_noise** command.

EXAMPLES

The following command forces NanoTime to compute noise beyond the rails:

```
nt_shell> set_noise_parameters -include_beyond_rails
```

SEE ALSO

```
report_noise(2)
si_enable_noise_analysis(3)
si_noise_margin_above_high(3)
si_noise_margin_above_low(3)
si_noise_margin_below_high(3)
si_noise_margin_below_low(3)
```

set_non_transparent

Sets non-transparent checking behavior on specified topologies or clocks in the design.

SYNTAX

```
status set_non_transparent
   [-edge_triggered]
   object_list
```

Data Types

```
   object_list          list
```

ARGUMENTS

-edge_triggered

Assign edge-triggered nontransparency, as in the case of a flip-flop, to the specified topology.

object_list

Specifies the topologies and clocks affected by the command.

DESCRIPTION

This command sets non-transparent checking behavior on specified topologies or clocks in the design. When this type of checking is set on a latch or a precharge net, path tracing cannot continue past that object. When it is set on a clock, NanoTime uses non-transparent checking behavior on all latches clocked by that clock.

To cancel the effects of the **set_non_transparent** command, use the **remove_non_transparent** command.

COMMAND PHASING

The **set_non_transparent** command must be executed after the **link_design** command but before the **trace_paths** or **extract_model** command.

EXAMPLES

The following command sets non-transparent checking on the latch that has its latch node at net n1.

```
nt_shell> set_non_transparent [get_topology \
          -of_object n1 -structure_type latch]
```

SEE ALSO

remove_non_transparent(2)

set_nonlinear_waveform

Specifies the type of transition waveform that NanoTime uses for analysis at specified nets in the design.

SYNTAX

```
status set_nonlinear_waveform
   [-samples number]
   [-threshold value]
   [-mode mode_string]
   net_list
```

Data Types

```
   number          integer
   value           float
   mode_string     string
   net_list        list
```

ARGUMENTS

-samples *number*

Specifies the number of data samples used to represent a nonlinear transition waveform, an integer from 5 to 100. A larger number results in a more accurate nonlinear model, at the cost of more memory and runtime. In the absence of this option, the number of samples is determined by the **nonlinear_waveform_samples** variable, which is set to 11 by default.

-threshold *value*

Specifies the threshold used to determine whether waveforms are smooth or nonlinear. The threshold must be a value between 1.0 and 10000. The lower the threshold, the more likely it is that a waveform is considered to be nonlinear. In the absence of this option, the threshold is determined by the **nonlinear_waveform_detection_ratio** variable, which is set to 5.0 by default.

-mode *mode_string*

Specifies the transition simulation mode. Valid values are as follows: **fast**, **detect_only**, **efficient**, or **accurate** (the default).

net_list

Specifies the list of nets affected by the command.

DESCRIPTION

By default, NanoTime converts each input transition waveform to a linear waveform for simulation of a stage. The linear slope approximation typically provides good tradeoff between accuracy and runtime. However, if your input transition waveforms differ significantly from the linear model, you can get better accuracy by using a nonlinear waveform for simulation, at the cost of memory and runtime. A nonlinear waveform uses a number of voltage and time data points rather than just two data points at the two threshold voltages.

NanoTime usually models an input transition as a simple linear slope from the lower threshold voltage to the upper threshold voltage for a rising transition, or from the upper to the lower threshold for a falling transition. The transition time is the amount of time it takes for the voltage to change from one threshold to the other.

The default thresholds are determined by the following variables:

```
rc_slew_lower_threshold_pct_fall
rc_slew_lower_threshold_pct_rise
rc_slew_upper_threshold_pct_fall
rc_slew_upper_threshold_pct_rise
```

By default, the lower thresholds are set to 10 and the upper thresholds are set to 90, so NanoTime models a transition as a straight line from 10 percent to 90 percent of the rail voltage. You can set thresholds for specific objects by using the **set_measurement_threshold** command.

NanoTime uses an algorithm to determine the smoothness of each calculated waveform. If the waveform is relatively smooth, the tool converts the waveform to an equivalent linear transition for simulating the next stage. On the other hand, if a waveform is relatively nonlinear, the tool can optionally use the piecewise linear (nonlinear) model for better accuracy.

The algorithm checks the smoothness by comparing the slopes of adjacent piecewise linear segments of the waveform. If the adjacent slopes are nearly the same, the waveform is considered smooth. If the slopes are sufficiently different, the waveform is considered nonlinear.

To perform this test, NanoTime considers the slopes of adjacent segments, here named A and B. The larger of the two slope ratios A/B and its inverse B/A is compared to the threshold value (5.0 by default). A ratio smaller than the threshold indicates a smooth waveform; a larger ratio indicates a nonlinear waveform. A glitch in the waveform might cause the ratio A/B to have a negative value. In that case, the waveform is considered to be nonlinear.

The comparison threshold can be set with the **-threshold** option to any 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. In the absence of this option, the threshold is determined by the **nonlinear_waveform_detection_ratio** variable, which is set to 5.0 by default.

When NanoTime saves the nonlinear transition waveform, the number of data points is the value specified by the **-samples** option. If the **-samples** option is not set for a net, the number of samples is determined by the **nonlinear_waveform_samples** variable, which is set to 11 by default. More samples provide better accuracy, at the cost of memory and runtime.

Modeling Modes

The **set_nonlinear_waveform** command specifies the scope of transition waveform modeling. The **-mode** option specifies one of four waveform handling modes: **fast**, **detect_only**, **efficient**, or **accurate** (the default). The modes operate as follows:

- The **fast** mode uses the simple linear model unconditionally. This mode is the fastest, but it can be inaccurate when the actual transition waveforms are significantly nonlinear. The **-threshold** option has no effect for this mode.
- The **detect_only** mode also uses the simple linear model. However, NanoTime runs the smoothness algorithm and marks the nets where the slope ratio exceeds the threshold. The nets that have been marked can be found by using the **report_net** command or by reading the **is_nonlinear** net attribute with the **get_attribute** command.
- The **efficient** mode runs the waveform smoothness algorithm and retains the nonlinear model where the slope ratio threshold is exceeded, and uses the linear waveform otherwise. This mode is recommended for initial analysis of new designs.
- The **accurate** mode always saves the original transition waveform at the net, ensuring the highest possible accuracy, at the cost of more runtime. The **-threshold** option has no effect for this mode.

Attributes

Transition simulation information is stored in three net attributes: **nonlinear_mode**, **has_waveform**, and **is_nonlinear**.

- The **nonlinear_mode** attribute is a string indicating the transition simulation mode for the net: **fast**, **detect_only**, **efficient**, or **accurate**.
- The **has_waveform** attribute is **true** when the nonlinear waveform data points are saved for the net, which happens during path tracing when the specified conditions are met.
- The **is_nonlinear** attribute is **true** when NanoTime determines that the slope ratio threshold is exceeded. This also happens during path tracing.

  You can get information about nets either by reading the attributes directly with the **get_attribute** command or by using the **report_net** command. In the report, the letter w in the Attrs column indicates that the **has_waveform** attribute is **true** and the letter n indicates that the **is_nonlinear** attribute is **true**.

COMMAND PHASING

The **set_nonlinear_waveform** command can be executed any time after the **link_design** command and before path tracing is initiated. If you execute the command after path tracing or model extraction is complete, NanoTime transitions back to the "design-checked" state and discards all operations performed after path tracing or model extraction.

EXAMPLES

The following command sets the transition simulation mode to **accurate** and sets the number of samples to 20 for net Sn32. The waveform at net Sn32 is saved as a set of 20 data samples and is used for simulation at the input of the stage at that net.

```
nt_shell> set_nonlinear_waveform -mode accurate \
          -samples 20 [get_nets Sn32]
```

The following command sets the transition simulation mode to **fast** for net Sn32. This cancels any previous mode set on that net and causes the linear waveform to be used for simulation at the input of the stage at that net.

```
nt_shell> set_nonlinear_waveform -mode fast \
          [get_nets Sn32]
```

SEE ALSO

check_design(2)
get_attribute(2)
report_net(2)
set_measurement_threshold(2)
trace_paths(2)
nonlinear_waveform_detection_ratio(3)
nonlinear_waveform_samples(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)

set_output_delay

Specifies the data required time at an output relative to a clock.

SYNTAX

```
status set_output_delay
   [-clock clock_name]
   [-clock_fall]
   [-level_sensitive]
   [-rise]
   [-fall]
   [-max]
   [-min]
   [-add_delay]
   [-network_latency_included]
   [-source_latency_included]
   delay_value
   port_pin_list
```

Data Types

```
   clock_name          list
   delay_value         float
   port_pin_list       list
```

ARGUMENTS

-clock *clock_name*

Specifies the name of a clock to which the specified delay is to be related. By default, if no clock is specified, for combinational designs the delay is relative to time zero. For sequential designs, the delay is relative to a new clock with a period determined by considering the sequential elements in the transitive fanout of each port.

-clock_fall

Specifies that the delay is relative to the falling edge of the clock specified by the **-clock** option, rather than the rising edge of that clock.

-level_sensitive

Indicates that the destination of the delay is a level-sensitive latch. This allows NanoTime to derive

setup and hold relationships for paths to this port as if it were a level-sensitive latch. If you do not use the **-level_sensitive** option, the output delay is treated as if it is a path to a flip-flop.

-rise

Indicates that *delay_value* refers to a rising transition on specified ports of the current design. If you do not specify the **-rise** or **-fall** option, rising and falling delays are assumed to be equal.

-fall

Specifies that *delay_value* refers to a falling transition on specified ports of the current design.

-max

Specifies that *delay_value* refers to the longest path. If you do not specify **-max** or **-min**, maximum and minimum output delays are assumed to be equal.

-min

Specifies that *delay_value* refers to the shortest path.

-add_delay

Adds the delay information to the existing output delay, instead of overwriting it. Use the **-add_delay** option to capture information about multiple paths leading from an output port that are relative to different clocks or different edges of the same clock.

-network_latency_included

Specifies whether the clock network latency should be included in the output delay value for an ideal clock. If this option is not specified, the clock network latency of the related clock is added to the output delay value. It has no effect if the clock is propagated or the output delay is not specified with respect to any clock.

-source_latency_included

Specifies whether to include the clock source latency in the output delay value. If this option is not specified, the clock source latency of the related clock is added to the output delay value. It has no effect if the output delay is not specified with respect to any clock.

delay_value

Specifies the amount of time before a related clock edge that a signal is required to be valid at an output, in time units of the technology. This usually represents a combinational path delay from the output to an external sequential element.

port_pin_list

Specifies the names of output ports or internal pins in the current design to which *delay_value* is assigned. If more than one object is specified, the objects must be enclosed in braces ({}).

DESCRIPTION

This command sets output path delay values for the current design. This is the amount of time it takes for a signal to travel from the output to the external sequential element that captures the data on a clock edge. If you do not specify an output delay for a port, NanoTimes assumes a delay of zero. The input and output delays characterize the operating environment of the current design, together with the **set_load** and **set_drive** commands.

Use the **-rise** or **-fall** option to restrict the output delay setting to rising or falling edges only at the specified ports. Use the **-min** or **-max** option to specify different worst-case minimum and maximum output delays.

The **set_output_delay** command sets the amount of delay from an output port to an external capture element relative to a clock edge. The clock is specified with the **-clock** option. Use the **-clock_fall** option if the delay is relative to the falling edge rather than the rising edge of the clock.

If the external capture element is a level-sensitive latch, use the **-level_sensitive** option. In that case, when NanoTime performs setup and hold checking, it assumes that the specified output delay applies to the capture of data on the closing edge of the clock, rather than the opening edge.

You can use the **-add_delay** option to specify output delays from a port relative to different clocks, or different edges of the same clock (with and without the **-clock_fall** option). First use a **set_output_delay** command to specify the delay with respect to the one clock, then use additional **set_output_delay** commands with the **-add_delay** option to specify the delays relative to other clocks. NanoTime then stores the delay information separately for each clock edge, instead of overwriting existing delay values.

To list output delays of internal pins, use the **report_paths** command.

To remove output delay values, use the **remove_output_delay** or **reset_design** command.

NOTE: The **set_input_delay** command is a Synopsys Design Constraints (SDC) command. SDC command options not listed in this man page are not supported by NanoTime. SDC commands having the same name might operate differently in different tools.

COMMAND PHASING

The **set_output_delay** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_output_delay** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following example sets an output delay of 1.7 relative to the rising edge of CLK1 for all output ports in the design.

```
nt_shell> set_output_delay 1.7 -clock CLK1 [all_outputs]
```

The following example sets the input and output delays for the bidirectional port INOUT1. The input signal arrives at INOUT1 2.5 units after the falling edge of CLK1. The output signal is required at INOUT1 at 1.4

units before the rising edge of CLK2.

```
nt_shell> set_input_delay 2.5 -clock CLK1 -clock_fall {INOUT1}

nt_shell> set_output_delay 1.4 -clock CLK2 {INOUT1}
```

The following example models the situation where there are three paths from output port OUT1. One of the paths is relative to the rising edge of CLK1. Another path is relative to the falling edge of CLK1. The third path is relative to the falling edge of CLK2. The **-add_delay** option adds the new output delay information without overwriting the old information.

```
nt_shell> set_output_delay 2.2 -max \
          -clock CLK1 -add_delay {OUT1}

nt_shell> set_output_delay 1.7 -max \
          -clock CLK1 -clock_fall -add_delay {OUT1}

nt_shell> set_output_delay 4.3 -max \
          -clock CLK2 -clock_fall -add_delay {OUT1}
```

To get a report on output delays that have been set, use the **report_port -output_delay** command. For example:

```
nt_shell> set_output_delay -clock MCLK -clock_fall \
          -min 0.10 [get_ports BUS[0]]
1
nt_shell> set_output_delay -clock MCLK -clock_fall \
          -max 0.12 [get_ports BUS[0]]
1
nt_shell> report_port -output_delay BUS[0]
 ...
Output Delay

Output Port Min Rise Min Fall Max Rise Max Fall Related Clock
----------- -------- -------- -------- -------- ----------------
BUS[0]         0.100    0.100    0.120    0.120 MCLK(f)
```

SEE ALSO

all_outputs(2)
create_clock(2)
current_design(2)
remove_output_delay(2)
report_design(2)
report_port(2)
reset_design(2)
set_drive(2)
set_load(2)
set_max_delay(2)
set_input_delay(2)

set_pbsa_override

Allows overriding of factors used in PBSA calculation with user-supplied values.

SYNTAX

```
status set_pbsa_override
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list
      | -rise_to rise_to_list
      | -fall_to fall_to_list]
   [-consecutive]
   [-use_existing_timing_points]
   [-Amax_factor factor]
   [-Amin_factor factor]
   [-Bmax_factor factor]
   [-Bmin_factor factor]
   [-Cmax_factor factor]
   [-Cmin_factor factor]
   [-Dmax_factor factor]
   [-Dmin_factor factor]
```

Data Types

```
   from_list           list
   rise_from_list      list
   fall_from_list      list
   through_list        list
   rise_through_list   list
   fall_through_list   list
   to_list             list
   rise_to_list        list
   fall_to_list        list
   factor              float
```

ARGUMENTS

-setup

Indicates that setup (maximum) paths are to be affected.

-hold

Indicates that hold (minimum) paths are to be affected.

-rise

Indicates that rising delays are to be affected, as measured at the path endpoint.

-fall

Indicates that falling delays are to be affected, as measured at the path endpoint.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all sequential elements and primary inputs related to that clock are used as path startpoints. If you specify a cell, all input pins and bidirectional pins of that cell are affected. You can use at most one of the **-from**, **-rise_from**, or **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only rising-edge signals from the specified objects are affected. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from ***fall_from_list***

Same as the **-rise_from** option, except that only falling-edge signals from the specified objects are affected.

-through ***through_list***

Specifies a list of pins, ports, cells, or nets through which the paths must pass. You can specify **-through** more than once in one command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net. If you do not specify any type of **through** option, all timing paths specified using the **-from** and **-to** options are affected.

-rise_through ***rise_through_list***

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through ***fall_through_list***

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all sequential elements and primary outputs related to that clock are used as path endpoints. If you specify a cell, all input pins and bidirectional pins of that cell are affected. You can use at most one of the **-to**, **-rise_to**, or **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge signals to the specified objects are affected. If a clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edges of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to ***fall_to_list***

Same as the **-rise_to** option, except that only falling-edge signals to the specified objects are affected.

-consecutive

Specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

-use_existing_timing_points

When this option is used the exception definition must only refer to pins which are already timing points. This can only be true after check_design has been run and the is_timing_vertex attribute is true for the needed pins. When this option is used then check_design does not need to be run again after the command has been issued.

-Amax_factor

Applies to the longest path in segment A (from the clock input port to the common point) under maximum operating conditions. Overrides the scaling factors KAmax and KALmax used in PBSA calculations.

-Amin_factor

Applies to the shortest path in segment A under minimum operating conditions. Overrides the scaling factors KAmin and KALmin used in PBSA calculations.

-Bmax_factor

Applies to the longest path in segment B (from the common point to the clock pin of the launch device) under maximum operating conditions. Overrides the scaling factors KBmax and KBLmax used in PBSA calculations.

-Bmin_factor

Applies to the shortest path in segment B under minimum operating conditions. Overrides the scaling factors KBmin and KBLmin used in PBSA calculations.

-Cmax_factor

Applies to the longest path in segment C (from the common point to the clock pin of the capture device) under maximum operating conditions. Overrides the scaling factors KCmax and KCLmax used in

PBSA calculations.

-Cmin_factor

Applies to the shortest path in segment C under minimum operating conditions. Overrides the scaling factors KCmin and KCLmin used in PBSA calculations.

-Dmax_factor

Applies to the longest path in segment D (from the clock pin of the launch device to the data input pin of the capture device) under maximum operating conditions. Overrides the scaling factors KDmax and KDLmax used in PBSA calculations.

-Dmin_factor

Applies to the shortest path in segment D under minimum operating conditions. Overrides the scaling factors KDmin and KDLmin used in PBSA calculations.

DESCRIPTION

You can invoke path-based slack adjustment (PBSA) to adjust the worst-case delay of each path segment by a specified amount. Compared with using the **set_clock_uncertainty** command to set global uncertainty values, using PBSA results in less pessimism, but requires more effort and runtime.

The **set_pbsa_override** command allows overriding the global path based slack adjustment multiplier values (pbsa_KAmax, pbsa_KAmin, etc.) for specified paths. Using the global PBSA factors, PBSA is calculated for segment A as:

delay = original_delay + pbsa_KAmax + pbsa_KALmax * path_length

However, if the **set_pbsa_override** command is used, such as in the command

set_pbsa_override -Amax_factor 0.05 -from x1.x2.clk

then for the specified paths the path-based calculated delay is overridden and the new delay is calculated to be the original delay multiplied by (1 + Amax_factor):

new_delay = original_delay * 1.05

For more information about path-based slack adjustment, see the NanoTime User Guide.

COMMAND PHASING

The **set_pbsa_override** command can only be executed after the **check_topology** command and before the **check_design** command.

SEE ALSO

remove_pbsa_override(2)
report_exceptions(2)
get_timing_paths(2)
report_paths(2)
set_clock_uncertainty(2)
pbsa_KALmax(3)
pbsa_KAmax(3)
pbsa_KBLmax(3)
pbsa_KBmax(3)
pbsa_KCLmax(3)
pbsa_KCmax(3)
pbsa_KDLmax(3)
pbsa_KDmax(3)
pbsa_KUhold(3)
pbsa_KUsetup(3)

set_phasing

Specifies same-cycle or next-cycle checking for instances of latches, precharge gates, and output ports in the design.

SYNTAX

```
status set_phasing
   [-setup]
   [-hold]
   [-same_cycle | -next_cycle]
   object_list
```

Data Types

```
   object_list      list
```

ARGUMENTS

-setup

Sets phase checking for setup checks only. In the absence of the **-setup** and **-hold** options, same-phase checking applies to both setup and hold checks.

-hold

Sets phase checking for hold checks only.

-same_cycle

Selects same-cycle checking for the listed objects.

-next_cycle

Selects next-cycle checking for the listed objects.

object_list

Specifies the latches, precharge gates, and output ports that are marked as requiring same-cycle or next-cycle checking.

DESCRIPTION

The **set_phasing** command selects same-cycle or next-cycle checking for specified instances of objects in the design where sequential timing checks are performed, such as latches, flip-flops, precharge gates, and output ports. This setting overrides the default behavior set with the **timing_default_phasing** variable, or any previous setting made on the same objects with the **set_phasing** command.

The **set_phasing** command is an object-based timing exception command. Object-based timing exception commands have lower priority than path-based timing exceptions specified with the **set_same_phase_path**, **set_no_same_phase_path**, **set_multicycle_path**, **set_false_path**, or **set_no_timing_check_path** commands.

Using the **set_phasing** command sets the **phasing** attribute on the object to **same_cycle** or **next_cycle**. To report the attributes that have been set on an object, use the **report_attribute** or **get_attribute** command. Some reporting commands such as the **report_port** command show the object-based phasing setting.

To reverse the effects of the **set_phasing** command, use the **remove_phasing** command.

How to Specify Phasing

There are several ways to specify same-phase or no-same-phase checking, listed here in order from highest to lowest priority:

- Use the **set_same_phase_path** and **set_no_same_phase_path** commands to define the behavior for specific objects.
- Use the **set_phasing** command to specify the behavior for instances of latches, precharge gates, and output ports.
- Use the **set_timing_check_attributes** command with the **-force_same_phase** or **-force_no_same_phase** option to define the behavior for specific timing checks.
- Use the **timing_default_phasing** variable, which specifies the behavior of all latches, precharge gates, and output ports when those objects are created or defined.

COMMAND PHASING

The **set_phasing** command is typically used after the **link_design** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command sets next-cycle checking on output port out1, for both setup and hold checking.

```
nt_shell> set_phasing -next_cycle [get_ports out1]
1
```

The following command sets same-cycle checking on the topology called latch1.

```
nt_shell> set_phasing -same_cycle [get_topology \
          -all -structure_type latch \
          -filter "structure_name == latch1"]
```

The following command sets same-cycle checking on all latches in the design.

```
nt_shell> set_phasing -same_cycle [get_topology \
          -all -structure_type latch]
```

SEE ALSO

get_attribute(2)
get_topology(2)
remove_no_same_phase_path(2)
remove_phasing(2)
remove_same_phase_path(2)
set_no_same_phase_path(2)
set_same_phase_path(2)
timing_default_phasing(3)
timing_intersection_transparency(3)

set_pin_variation

Modifies the POCV variation factor applied to the delays associated with the specified trigger objects.

SYNTAX

```
status set_pin_variation
   [-min]
   [-max]
   [-rise]
   [-fall]
   [-voltage supply_voltage]
   [-target target_objects]
   -variation variation_factor
   trigger_objects
```

Data Types

```
   variation_factor        float
   supply_voltage          float
   target_objects          list
   trigger_objects         list
```

ARGUMENTS

-min

Modifies variation factor for short path analysis.

-max

Modifies variation factor for long path analysis.

-rise

Modifies variation factor for rising triggers.

-fall

Modifies variation factor for falling triggers.

-variation *variation_factor*

The new variation factor, a positive number. This is a 1 sigma variation scale factor that is used with

the original calculated delay and sigma value specified to determine the stage variation.

-voltage *supply_voltage*

The voltage value to which this variation override applies. If a supply is specified with a variation override, the override will only apply if this voltage value matches the voltage at the stage of interest. Multiple overrides with different voltage values may be applied at a single trigger pin. If an override is defined without a voltage specified, it will apply to all voltages. If a voltage specific and a non-voltage specific override have been defined, the voltage specific value will be used.

-target *target_objects*

The list of nets, transistor gate pins, or output ports affected by the command. This list defines a set of delay calculation target pins, nets, or ports. Trigger objects must be specified. If target objects are specified in the same command, the variation factor for the trigger-target pair only is modified. A trigger-target pair specification will take precedence over a trigger only specification.

trigger_objects

The list of trigger nets or transistor gate pins affected by the command. This list defines a set of delay calculation trigger pins or nets. The variation factors applied to the delay values from these trigger pins or nets to a driven output net are modified.

DESCRIPTION

The **set_pin_variation** command replaces the calculated variation factor at the specified trigger points in the design. The user defined variation value is a 1 sigma variation scale factor that is applied to the calculated delay. Variation calculations use a num_sigma value defined by the variables timing_pocv_sigma or timing_pocv_sigma_min. At each specified trigger net or transistor gate, NanoTime modifies the variation calculated as follows:

```
variation = (calculated_delay * variation_factor * num_sigma)
```

This feature can be used to adjust variation values at inputs of transistor based stages, dds/dcs input pins, or .lib model input pins.

In the command, you must specify the types of timing arcs affected by the command (min, max, rise, fall) and a list of objects that define which arcs to apply new variation values. You specify the affected arcs with a list of trigger objects as an argument to the command and with an optional list of target objects as an argument to the **-target** option.

To remove pin variation values that have been set, use the **remove_pin_variation** command. To report pin variation values that have been set, use the **report_pin_variation** command.

COMMAND PHASING

The **set_pin_variation** command is typically executed in the "netlist-linked" or "topology-checked" states (in other words, between the **link_design** command and the **check_design** command). The command cannot be executed before the **link_design** command. If you use the **set_pin_variation** command after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command modifies the variation calculation for arcs starting at pin X1.XP1.G for max-path delay variation calculation.

```
nt_shell> set_pin_variation -max \
          -variation 0.02 [get_pins X1.XP1.G ]
```

NanoTime first calculates the delay from pin X1.XP1.G, then multiplies the result by 0.02, then multiplies by the number of sigma specified in timing_pocv_sigma. If the calculated delay is 4.0, and timing_pocv_sigma is 3.0, the variation is calculated as follows:

```
stage variation = (4.0 * 0.02 * 3.0)
```

SEE ALSO

```
report_pin_variation(2)
remove_pin_variation(2)
set_variation_parameters(2)
timing_pocv_sigma(3)
timing_pocv_sigma_min(3)
```

set_port_direction

Sets the signal flow direction of specified ports.

SYNTAX

```
status set_port_direction
   { -input | -output | -inout }
   port_list
```

Data Types

```
   port_list         list
```

ARGUMENTS

-input

Specifies that the listed ports are input ports.

-output

Specifies that the listed ports are output ports.

-inout

Specifies that the listed ports are bidirectional ports.

port_list

The list of ports affected by the command.

DESCRIPTION

The **set_port_direction** command identifies the signal flow direction of one or more ports. This is necessary when the port direction information is not available from the netlist.

COMMAND PHASING

The **set_port_direction** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command specifies that the ports with names beginning with "in" are input ports.

```
nt_shell> set_port_direction -input in*
```

SEE ALSO

create_port(2)
register_netlist(2)

set_precharge_input_net

Specifies that a net is driven only by precharge logic.

SYNTAX

```
status set_precharge_input_net
   net_list
```

Data Types

net_list	list

ARGUMENTS

net_list

Specifies the nets that are driven only by precharge logic.

DESCRIPTION

This command specifies that a net in the design is driven only by precharge logic. You must specify one or more nets in the design.

The types of timing checks that NanoTime performs on a precharge gate depend on whether the inputs are driven by combinational logic or precharge logic. When you specify that an input net is a precharge input net, NanoTime can avoid the timing checks for combinational logic.

To cancel the effects of the **set_precharge_input_net** command, use the **remove_precharge_input_net** command.

COMMAND PHASING

The **set_precharge_input_net** command is typically executed in the "netlist-linked" or "topology-checked" state (in other words, between the **link_design** command and the **check_design**

command). If you execute the **set_precharge_input_net** commani after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all operations performed after the **check_design** command.

EXAMPLES

The following command specifies that net n1 is a precharge input net.

```
nt_shell> set_precharge_input_net [get_nets n1]
```

SEE ALSO

remove_precharge_input_net(2)

set_propagated_clock

Specifies propagated clock latency (rather than ideal clocking) for specified objects.

SYNTAX

status **set_propagated_clock** *object_list*

Data Types

object_list list

ARGUMENTS

object_list

A list of clocks, ports, or pins that are to use propagated (rather than ideal) clock latency. If you specify a clock, the whole clock network uses propagated latency. If you specify a port or pin, propagated clock latency is used for all objects in the transitive fanout of the specified port or pin.

DESCRIPTION

The **set_propagated_clock** command causes specified clocks, ports, or pins to used propagated (not ideal) clocking, thereby canceling the effects of the **set_clock_latency** or **remove_propagated_clock** command. Propagated clocking is the default behavior in NanoTime.

For each sequential element in the design, NanoTime uses either propagated or ideal clocking. 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.

To change a clock, port, or pin to use ideal clocking, use the **remove_propagated_clock** command or use the **set_clock_latency** command to specify an ideal latency value. To restore propagated clocking behavior, use the **set_propagated_clock** command.

To see propagated clock attributes on clocks, use the **report_clock** command. For information about propagated clock attributes on clocks, see the man page for the **report_clock** command.

For more information about propagated versus ideal clocking, see the man page for the **set_clock_latency** command.

COMMAND PHASING

The **set_transition_coefficients** command is typically executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **set_transition_coefficients** command after the **check_design** command, the NanoTime tool transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command makes all clocks in the design use propagated clocking, thereby canceling the effects of any **set_clock_latency** commands applied to clocks.

```
nt_shell> set_propagated_clock [all_clocks]
```

SEE ALSO

```
remove_propagated_clock(2)
report_clock(2)
set_clock_latency(2)
```

set_requires_clock

Sets the is_requires_clock attribute.

SYNTAX

```
status set_requires_clock
   object_list
```

Data Types

```
   object_list       collection
```

ARGUMENTS

object_list

The list or collection of ports, nets and pins to be modified.

DESCRIPTION

This command sets the **is_requires_clock** attribute on a list of ports, nets, or pins. You can use this capability to debug the clock network by setting the attribute on objects that you know to be part of the clock network. When you run the **report_clock_network** command, NanoTime reports errors for nets that have the **is_requires_clock** attribute but that the tool has determined are not part of the clock network.

COMMAND PHASING

The **set_requires_clock** command is typically executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command).

You cannot use the command before the **link_design** command. If you use the **set_requires_clock** command after the **check_topology** command, the NanoTime tool transitions back to the "netlist-

linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following example sets the **is_requires_clock** attribute on net latch_clock.

```
nt_shell> set_requires_clock "latch_clock"
```

SEE ALSO

report_clock_network(2)
remove_requires_clock(2)

set_same_phase_path

Sets same-phase checking on specified paths, causing NanoTime to check for data arrival in the same clock phase rather than the next clock phase.

SYNTAX

```
status set_same_phase_path
   [-setup]
   [-hold]
   [-rise]
   [-fall]
   [-from from_list
      | -rise_from rise_from_list
      | -fall_from fall_from_list]
   [-through through_list]
   [-rise_through rise_through_list]
   [-fall_through fall_through_list]
   [-to to_list]
      | -rise_to rise_to_list]
      | -fall_to fall_to_list]
   [-consecutive]
   [-use_existing_timing_points]
```

Data Types

```
   from_list           list
   rise_from_list      list
   fall_from_list      list
   through_list        list
   rise_through_list   list
   fall_through_list   list
   to_list             list
   rise_to_list        list
   fall_to_list        list
```

ARGUMENTS

-setup

Sets same-phase checking for setup checks only. In the absence of the **-setup** and **-hold** options, same-phase checking applies to both setup and hold checks.

-hold

Sets same-phase checking for hold checks only.

-rise

Specifies that only paths with a rising transition at the path endpoint are affected. In the absence of any of the **-rise**, **-fall**, or related options, both rising and falling edges are affected.

-fall

Specifies that only paths with a falling transition at the path endpoint are affected.

-from ***from_list***

Specifies a list of timing path startpoint objects. A valid timing startpoint is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell, a data pin of a level-sensitive latch, or a pin that has input delay specified. If a clock is specified, all sequential elements and primary inputs related to that clock are used as path startpoints. If you specify a cell, all input pins and bidirectional pins of the cell are used as path startpoints. You can use at most one of the **-from**, **-rise_from**, and **-fall_from** options.

-rise_from ***rise_from_list***

Same as the **-from** option, except that only rising-edge signals from the specified objects are affected. If a clock object is specified, this option selects startpoints clocked by the named clock, but only the paths launched by rising edge of the clock at the clock source, taking into account any logical inversions along the clock path.

-fall_from ***fall_from_list***

Same as the **-rise_from** option, except that only falling-edge signals from the specified objects are affected.

-through ***through_list***

Specifies a list of pins, ports, cells, or nets through which the paths must pass. You can use the **-through** option more than one time in one command invocation. Nets are interpreted to imply the leaf-level fanout pins of the net.

-rise_through ***rise_through_list***

The same as the **-through** option, but only affects paths with rising-edge transitions arriving at the specified objects.

-fall_through ***fall_through_list***

The same as the **-through** option, but only affects paths with falling-edge transitions arriving at the specified objects.

-to ***to_list***

Specifies a list of timing path endpoint objects. A valid timing endpoint is a clock, a primary output or inout port, a sequential cell, a data pin of a sequential cell, or a pin that has output delay specified. If a clock is specified, all registers and primary outputs related to that clock are used as path endpoints. If a cell is specified, all input pins and bidirectional pins of the cell are used as path endpoints. You can use at most one of the **-to**, **-rise_to**, and **-fall_to** options.

-rise_to ***rise_to_list***

Same as the **-to** option, except that only rising-edge signals to the specified objects are affected. If a

clock object is specified, this option selects endpoints clocked by the named clock, but only the paths captured by rising edges of the clock at clock source, taking into account any logical inversions along the clock path.

-fall_to ***fall_to_list***

Same as the **-rise_to** option, except that only falling-edge signals to the specified objects are affected.

-consecutive

Specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

-use_existing_timing_points

When this option is used the path definition must only refer to pins which are already timing points. This can only be true after the **check_design** command has been run and the **is_timing_vertex** attribute is **true** for the needed pins. When this option is used, then the **check_design** command does not need to be run again after the **set_same_phase_path** command has been issued.

DESCRIPTION

The **set_same_phase_path** command specifies same-phase checking for all timing paths that meet the specified the criteria in the **-from**, **-through**, and **-to** options.

This command defines a path-based timing exception, along with the **set_no_same_phase_path**, **set_multicycle_path**, **set_false_path**, **set_no_timing_check_path**, and **set_exclude_reference_path** commands. Path-based timing exceptions have priority over object-based timing exceptions, such as those set with the **set_phasing** or **set_timing_check_attributes** commands or invoked via the **timing_default_phasing** variable.

In case of conflicting exceptions for a path, the timing exception types have the following order of priority, from highest to lowest:

```
1. set_false_path
2. set_no_timing_check_path
3. set_multicycle_path
4. set_no_same_phase_path
5. set_same_phase_path
```

Multiple exception settings that are not in conflict with each other can apply to a single path. There is no conflict between the **set_multicycle_path** command and the **set_same_phase_path** or **set_no_same_phase_path** commands. Also, there is no conflict between the **-setup** and **-hold** settings for a specific exception.

To undo the effects of a **set_same_phase_path** command, use the **remove_same_phase_path** command.

Same-Phase and No-Same-Phase 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-phase 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 following clock cycle rather than the same clock cycle. This behavior can be changed by setting the **timing_intersection_transparency** variable.

Under same-phase 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 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.

Under same-phase checking conditions, if a downstream latch is designed to capture data in the next clock phase rather than the same clock phase, you need to set no-same-phase checking so that the timing checks are performed at the correct times. In 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.

How to Specify Phasing

There are several ways to specify same-phase or no-same-phase checking, listed here in order from highest to lowest priority:

- Use the **set_same_phase_path** and **set_no_same_phase_path** commands to define the behavior for specific objects.
- Use the **set_phasing** command to specify the behavior for instances of latches, precharge gates, and output ports.
- Use the **set_timing_check_attributes** command with the **-force_same_phase** or **-force_no_same_phase** option to define the behavior for specific timing checks.
- Use the **timing_default_phasing** variable, which specifies the behavior of all latches, precharge gates, and output ports when those objects are created or defined.

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** type options in the command specify the paths that are to use same-phase or no-same-phase checking, irrespective of the settings on objects in the paths.

You can use multiple **-through**, **-rise_through**, and **fall_through** options in a single command to specify paths that traverse multiple points in the design. If more than one object is specified within one **-through**, **-rise_through**, or **fall_through** option, the path can pass through any one of the listed objects. The **-consecutive** option specifies that the "from," "through," and "to" points must be consecutive, without other points in between.

COMMAND PHASING

The **set_same_phase_path** command is typically used after the **check_topology** command and before the **check_design** command. If you use it after the **check_design** command, NanoTime transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

However, if you use the **-use_existing_timing_points** option with the **set_same_phase_path** command, you can execute it after the **check_design** command and before the **trace_paths** command or **extract_model** command.

EXAMPLES

The following command sets same-phase checking for all timing paths from port in1 to port outd3.

```
nt_shell> set_same_phase_path \
          -from [get_ports in1] \
          -to [get_ports outd3]
```

SEE ALSO

remove_same_phase_path(2)
report_exceptions(2)
set_exclude_reference_path(2)
set_false_path(2)
set_multicycle_path(2)
set_no_same_phase_path(2)
set_no_timing_check_path(2)
timing_intersection_transparency(3)
timing_default_phasing(3)
set_phasing(2)
remove_phasing(2)

set_search_enabled

Enables the searching of specified lib_topology objects.

SYNTAX

```
status set_search_enabled
   [-only_used]
   [-from_file file_name]
   [lib_topology_list]
```

Data Types

file_name	string
lib_topology_list	list

ARGUMENTS

-only_used

Causes NanoTime to search for all loaded lib_topology objects, whether or not they were previously enabled for searching, and to mark each lib_topology object's search_enabled attribute to true if the topology exists in the design, or to false otherwise. No actions are carried out on the topologies found.

-from_file *file_name*

Causes NanoTime to enable searching for the lib_topology objects listed in a file. In the file, each line should have a topology library name followed by the lib_topology name, separated by a space character.

lib_topology_list

A collection of lib_topology objects created with the **get_lib_topology** command. NanoTime enables searching of the specified lib_topology objects.

DESCRIPTION

This command enables specified lib_topology objects for searching. The effect of the command is to set the search_enabled attribute of the specified lib_topology objects to true. When you later search for topologies with the **match_topology** or **check_topology** command, NanoTime searches only for the lib_topology objects that have their corresponding search_enabled attribute set to true.

In the **set_search_enabled** command, to specify the lib_topology objects explicitly, use the **get_lib_topology** command. In the **get_lib_topology** command, you specify the lib_topology objects in the following format:

library_name.pattern

where *library_name* is the name of the topology library, the period is the hierarchy separator character (which can be changed by setting the **hierarchy_separator** variable) and *pattern* is the lib_topology name or a wildcard pattern that matches one or more lib_topology names.

For example, to enable searching of the mux3 lib_topology in the user2 topology library:

```
nt_shell> set_search_enabled [get_lib_topology user2.mux3]
```

Instead of using the **get_lib_topology** command, you can list the lib_topology objects in separate file. For example,

```
nt_shell> set_search_enabled -from_file mylist
```

In the separate file "mylist," you specify each lib_topology object by library name and lib_topology name, separated by a space. For example,

```
common nando_istack2
common noro_istack2
global latch_itri_ostack
user2 mux3
 ...
```

Instead of specifying the lib_topology objects with the **get_lib_topology** command or providing a list, you can have NanoTime automatically search the design for all defined topologies and set the search_enabled flag to true for all lib_topology objects that exist in the design. Use the **-only_used** option:

```
nt_shell> set_search_enabled -only_used
```

This command performs a search for all loaded lib_topology objects, whether or not they were previously enabled for searching. It does not carry out any of the actions defined for the topology. Instead, it only marks each lib_topology object's search_enabled attribute to true if the topology exists in the design, or to false if it does not exist in the design. After running this command you can use the **list_lib_topology -only_used** command to list only the lib_topology objects that exist in the design.

The **remove_search_enabled** command has the opposite effect of the **set_search_enabled** command. It sets the search_enabled attribute to false for specified lib_topology objects.

COMMAND PHASING

The **set_search_enabled** command can be executed any time after the **link_design** command.

EXAMPLES

The following command enables searching of the mux3 lib_topology in the user2 topology library.

```
nt_shell> set_search_enabled [get_lib_topology user2.mux3]
```

The following command enables searching of the lib_topology objects listed in the file named "mylist."

```
nt_shell> set_search_enabled -from_file mylist
```

The topologies should be listed in the file "mylist," one entry per line, with the topology library name and lib_topology name separated by a space character.

The following command searches for all topologies, whether or not searching is enabled for them, and changes the search_enabled attribute to true for all lib_topology objects that exist in the design (and to false for those that don't) without performing any actions on the topologies found:

```
nt_shell> set_search_enabled -only_used
```

The following command creates a collection of global lib_topology objects for which searching is currently enabled:

```
nt_shell> get_lib_topology \
          -filter "search_enabled==true" global.*
```

SEE ALSO

check_topology(2)
get_lib_topology(2)
list_lib_topology(2)
match_topology(2)
read_topology_library(2)
remove_search_enabled(2)
hierarchy_separator(3)

set_si_delay_analysis

Sets reselection or arrival parameters on specified nets for crosstalk analysis.

SYNTAX

```
status set_si_delay_analysis
   [-reselect rnets]
   [-ignore_arrival inets]
   [-exclude]
   [-victims vnets]
      | -rise_victims rise_vnets
      | -fall_victims fall_vnets]
   [-aggressors anets]
   [-coupling_factor cfactor]
   [-min]
   [-max]
```

Data Types

```
   rnets     list
   inets     list
   vnets     list
   rise_vnets     list
   rise_vnets     list
   anets     list
   cfactor   float
```

ARGUMENTS

-reselect *rnets*

Specifies a list of nets to be reselected in each iteration, irrespective of reselection criteria. A net that has already been removed from consideration by filtering cannot be reselected.

-ignore_arrival *inets*

Specifies a list of nets to be analyzed using infinite arrival windows.

-exclude

Excludes consideration of specified nets as victims or aggressors, in combination with the **-victims** or **-**

aggressors options. When both the **-victims** and **-aggressors** options are used, all cross-coupling capacitors between the two sets of listed nets are reduced to grounded capacitors, one on each net and both with the same value as the coupling capacitor. The grounded capacitors are then multiplied by the coupling capacitance factor.

-victims ***vnets***

With the **-exclude** option, specifies a list of nets excluded from consideration as victim nets.

-rise_victims ***rise_vnets***

Same as the **-victims** option, except that exclusions only apply to rising edge transitions of the victim.

-fall_victims ***fall_vnets***

Same as the **-victims** option, except that exclusions only apply to falling edge transitions of the victim.

-aggressors ***anets***

With the **-exclude** option, specifies a list of nets excluded from consideration as aggressor nets.

-coupling_factor ***cfactor***

Sets the coupling capacitance multiplication factor to be used with the excluded nets defined by the **-exclude** option in the same command line. Allowable values are 0.0 to 2.0, inclusive. The default is 1.0.

-min

With the **-exclude** option, excludes nets from consideration for minimum-delay analysis only.

-max

With the **-exclude** option, excludes nets from consideration for maximum-delay analysis only.

DESCRIPTION

This command sets the reselection parameters explicitly on specified nets for crosstalk analysis, overriding the default reselection criteria on a net-by-net basis.

The **-reselect** option causes a specified list of nets to be reselected for all crosstalk iterations, irrespective of the default reselection criteria. Note that a net already removed from consideration by filtering cannot be reselected.

The **-ignore_arrival** option causes a specified list of nets to be analyzed using infinite arrival windows. For each net in the list considered as a victim, the arrival times of all coupled aggressors are ignored. For each net in the list considered as an aggressor, the arrival times on that aggressor are ignored. This results in a conservative (possibly pessimistic) crosstalk calculation.

The **-exclude** option excludes consideration of specified nets as victims or aggressors, in combination with the **-victims** or **-aggressors** options. The scope of exclusion can be limited by using the **-min** or **-max** option.

When you exclude nets from SI (crosstalk) analysis, all coupling capacitors related to those nets are split to ground and multiplied by a coupling capacitance factor. You can define default multiplier values for min and max analysis by setting the **si_exclusion_cap_factor_min** and **si_exclusion_cap_factor_max** variables. Allowable values are 0.0 to 2.0 inclusive and the default is 1.0. These values are used whenever the **set_si_delay_analysis -exclude** command is executed without a net-specific coupling factor.

To define a multiplication factor to use with a specific net, use the **-exclude** and **-coupling_factor** options at the same time. Allowable coupling factor values are 0.0 to 2.0 inclusive and the default is 1.0.

To get a report on the settings you have made, use the **report_si_delay_analysis** command.

To reverse the effects of the **set_si_delay_analysis** command, use the **remove_si_delay_analysis** command.

COMMAND PHASING

The **set_si_delay_analysis** command must be executed after the **check_design** command but before path tracing commands such as **trace_paths**, **extract_model**, or **characterize_context**. The **si_exclusion_cap_factor_min** and **si_exclusion_cap_factor_max** variables must be set before the **set_si_delay_analysis** command is executed.

EXAMPLES

The following command causes all nets *CLK_NET_** to be excluded from crosstalk analysis as victim nets.

```
nt_shell> set_si_delay_analysis -exclude -victims \
          [get_nets CLK_NET*]
```

The following command causes all nets *SCAN_LOGIC** to be reselected in all crosstalk analysis iterations.

```
nt_shell> set_si_delay_analysis -reselect \
          [get_nets SCAN_LOGIC*]
```

The following command causes all nets to be analyzed with infinite arrival windows.

```
nt_shell> set_si_delay_analysis -ignore_arrival \
          [get_nets *]
```

The following example ensures that the scan clock *SCN_CLK_** does not affect the main clock network *CLK_NET_**, and vice versa.

```
nt_shell> set_si_delay_analysis -exclude \
          -victims [get_nets SCN_CLK_*] \
          -aggressors [get_nets CLK_NET*]

nt_shell> set_si_delay_analysis -exclude \
          -victims [get_nets CLK_NET*]
          -aggressors [get_nets SCN_CLK_*]
```

The following command excludes nets AGGR1 and VICT1 from SI analysis. Assume the coupling factor is

1.0 (the default). If there is a 4pF coupling capacitor between the two nets, the exclusion command removes that capacitor and adds a 4pF capacitor to ground from each of the nets AGGR1 and VICT1.

```
nt_shell> set_si_delay_analysis -exclude -aggressors AGGR1 -victims VICT1
```

The following example sets different coupling factors for maximum and minimum analysis when excluding nets AGGR1 and VICT1. The original 4pF coupling capacitor is replaced by two 4.8pF grounded capacitors during maximum delay analysis (one on each net) and two 3.2pF grounded capacitors during minimum delay analysis (one on each net).

```
nt_shell> set_si_delay_analysis -exclude -aggressors AGGR1 \
           -victims VICT1 -max -coupling_factor 1.2
nt_shell> set_si_delay_analysis -exclude -aggressors AGGR1 \
           -victims VICT1 -min -coupling_factor 0.8
```

The following example sets factors to be used with all **set_si_delay_analysis -exclude** commands that do not include net-specific factors. The original 4pF coupling capacitor is replaced by two 8pF grounded capacitors during maximum delay analysis (one on each net) and zero capacitance during minimum delay analysis.

```
nt_shell> set si_exclusion_cap_factor_max 2.0
nt_shell> set si_exclusion_cap_factor_min 0.0

nt_shell> set_si_delay_analysis -exclude -aggressors AGGR1 -victims VICT1
```

SEE ALSO

```
remove_si_delay_analysis(2)
report_si_delay_analysis(2)
report_si_nets(2)
si_enable_analysis(3)
si_exclusion_cap_factor_max(3)
si_exclusion_cap_factor_min(3)
```

set_si_noise_analysis

Sets exceptions to default behavior on specified nets for noise analysis.

SYNTAX

```
status set_si_noise_analysis
   -exclude
   -victims vnets
   -aggressors anets
   [-above]
   [-below]
   [-high]
   [-low]
```

Data Types

```
   vnets          list
   anets          list
```

ARGUMENTS

-exclude

This option is required. Excludes consideration of specified nets as noise victims or aggressors, in combination with one or both of the **-victims** and **-aggressors** options.

-victims *vnets*

Specifies a list of nets excluded from consideration as noise victim nets. One or both of the **-victims** and **-aggressors** options must be used.

-aggressors *anets*

Specifies a list of nets excluded from consideration as noise aggressor nets. One or both of the **-victims** and **-aggressors** options must be used.

-above

Excludes the specified nets from noise analysis when the noise on the victim net is above the stable signal level (either high or low).

-below

Excludes the specified nets from noise analysis when the noise on the victim net is below the stable signal level (either high or low).

-high

Excludes the specified nets from noise analysis when the noise is relative to a high signal level on the victim net (i.e. the supply voltage).

-low

Excludes the specified nets from noise analysis when the noise is relative to a low signal level on the victim net (i.e. the ground voltage).

DESCRIPTION

This command defines exceptions to the default behavior when determining which nets should be considered for noise analysis. The **-exclude** option and at least one of the **-victims** and **-aggressors** options are required.

The **-exclude** option excludes consideration of specified nets as noise victims or aggressors, in combination with the **-victims** or **-aggressors** options.

The scope of exclusion can be limited by using the **-above**, **-below**, **-high**, or **-low** options alone or in combination. These options refer to the noise on the victim net. For example, the **-high** option means that when the victim is high, exclude the analysis of noise injection to the specified victim.

To get a report on the current settings, use the **report_si_noise_analysis** command.

To reverse the effects of the **set_si_noise_analysis** command, use the **remove_si_noise_analysis** command.

COMMAND PHASING

The **set_si_noise_analysis** command can be executed any time after the **check_design** command. It must be executed before the **update_noise** command to affect the noise analysis.

EXAMPLES

The following command causes all nets *CLK_NET_** to be excluded from noise analysis as victim nets.

```
nt_shell> set_si_noise_analysis -exclude -victims [get_nets CLK_NET*]
```

The following example specifies that NanoTime should not analyze the noise effects of the scan clock *SCN_CLK_** on the main clock network *CLK_NET_** and vice versa.

```
nt_shell> set_si_noise_analysis -exclude \
```

```
            -victims [get_nets SCN_CLK_*] \
            -aggressors [get_nets CLK_NET*]

nt_shell> set_si_noise_analysis -exclude \
            -victims [get_nets CLK_NET*]
            -aggressors [get_nets SCN_CLK_*]
```

The following command specifies that NanoTime should not analyze noise of types above high and above low on victim net Z1.

```
nt_shell> set_si_noise_analysis -exclude -above -victims Z1
```

SEE ALSO

```
remove_si_noise_analysis(2)
report_si_noise_analysis(2)
report_noise(2)
si_enable_noise_analysis(3)
```

set_simulation_attributes

Sets the operating parameters of a simulation object for use with dynamic delay simulation.

SYNTAX

```
status set_simulation_attributes
   [-control_header ch_file]
   [-spice_header sh_file]
   [-time simulation_time]
   [-vectors v_file]
   sim_object_list
```

Data Types

```
   ch_file            string
   sh_file            string
   simulation_time    float
   v_file             string
   sim_object_list    list
```

ARGUMENTS

-control_header *ch_file*

(Obsolete) Name of the control header file in the search path. The control header file contains the simulation control commands.

-spice_header *sh_file*

Name of the SPICE header file in the search path. The SPICE header file contains the SPICE control statements.

-time *simulation_time*

The total simulation time span, in design time units.

-vectors *v_file*

Name of the vector definition file in the search path. The vector definition file contains the sequence of input signals to the simulation region.

sim_object_list

A list of one or more simulation objects collected with the **all_simulations** or **get_simulations** command.

DESCRIPTION

The **set_simulation_attributes** command sets the operating parameters of a simulation object: the control header file name, the SPICE header file name, the total number of time units in the simulation run, the vector file name, and list of simulation objects affected by the command.

The **set_simulation_attributes** command is used after the **check_design** command, which creates the simulation regions, but before the **trace_paths** command, which runs the simulations. The simulation objects must be collected with the **all_simulations** or **get_simulations** command, typically by setting a variable equal to the collection. For example:

```
nt_shell> set mysim [get_simulations -input in1]
 ...
nt_shell> set_simulation_attributes
          -time 10000
          -vectors v1.vec $mysim
 ...
```

The **set_simulation_attributes** command can specify a SPICE header file with the **-spice_header** option. The models specified with the **read_spice_model** command have higher priority. This file specifies the SPICE models and *nanosim tech parameters that will be used in the dynamic simulation region. 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
```

The **set_simulation_attributes** command can specify a vector file with the **-vectors** option. In the absence of a vector file, NanoTime tries all possible combinations of input values. You can use a vector file to specify simultaneous switching of different inputs, or to reduce the number of vectors exercised in the simulation.

You can create a template vector file for a specific simulation region by using the **report_simulation -template** command. Edit the template file as needed. In a vector file, a semicolon at the beginning of a line denotes a comment. A template file might contain semicolons that must be removed to implement optional settings.

The following is a simple example of a vector file.

```
Outputs:
dec2_n

Signals:
ip0 ip1_n dec17
```

```
Vectors:
; in1_VALUE (DELAY_VALUE)  in2_VALUE (DELAY_VALUE)  in3_VALUE (DELAY_VALUE)
r 1 1
f 0 0
1 r 1
0 f 0

States:
net1 net2

Initial Conditions:
0.9 0
0 0.9

Options:
min max
```

The following list shows the allowed characters in the vector field. Stable expands into 0 and 1, and switching expands into r and f.

```
1 - constant 1
0 - constant 0
r - rising
f - falling
x - stable
* - switching
```

In addition to the vectors, you can optionally specify initial conditions (voltages) for a list of nets. Write the keyword **States:** on one line and the list of nets (separated by spaces) on the following line. Next, write the keyword **Initial Conditions:** on a new line, followed by a list of initial condition voltages (separated by spaces) on the next line. The number of initial condition values must equal the number of nets.

You can provide any number of sets of initial conditions for the same list of nets. However, you cannot provide a different list of nets. Write each set of initial conditions (voltages) on a separate line. All initial conditions must appear consecutively before any other keywords in the file.

A separate simulation is performed for every combination of input vectors and initial conditions.

COMMAND PHASING

The **set_simulation_attributes** command must be executed after the **check_design** command but before the **trace_paths** command.

EXAMPLES

The following command sets the simulation runtime to 10,000 picoseconds and the vector file to v1.vec for the simulation object mysim:

```
nt_shell> set_simulation_attributes
          -time 10000
          -vectors v1.vec $mysim
```

In this example, the design time units are picoseconds.

SEE ALSO

```
all_simulations(2)
check_design(2)
get_simulations(2)
mark_simulation(2)
report_simulation(2)
```

set_simulator

Uses an external simulator for delay calculation on the specified nets.

SYNTAX

```
status set_simulator
   [-ccb_containing]
   nets
```

Data Types

```
   nets        list
```

ARGUMENTS

-ccb_containing

Any simulation units derived from the channel-connected block containing the specified nets are analyzed with an external simulator.

-hspice

Uses the HSPICE tool as the external simulator.

-finesim_embedded

Uses the FineSim Embedded tool as the simulator.

nets

The net or nets included in the simulation region.

DESCRIPTION

The **set_simulator** command specifies a list of nets. Any timing arcs that contain the specified nets are analyzed with an external simulator. This provides enhanced accuracy at the cost of increased runtime.

The **-ccb_containing** option uses the external simulator for all possible timing paths through the channel-connected blocks that contain the nets in the argument list. The number of paths and therefore the runtime might increase greatly if you specify a large number of nets in the argument list.

By default, HSPICE is the external simulator, if neither the **-hspice** option nor the **-finesim_embedded** option is used.

COMMAND PHASING

The **set_simulator** command must be executed after the **check_topology** command but before the **check_design** command. If you use the **set_simulator** command after the **check_design** command, NanoTime discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command uses HSPICE to analyze the timing on all nets whose names begin with Xareg.A:

```
nt_shell> set_simulator  Xareg.A*
1
```

The following command uses FineSim Embedded to analyze the timing on all nets whose names begin with Xareg.A:

```
nt_shell> set_simulator -finesim_embedded Xareg.A*
1
```

set_soi_parameters

Sets the parameters required to determine body voltage ranges for SOI transistors.

SYNTAX

```
status set_soi_parameters
   [-model name]
   [-conservative]
   [-average]
   [-scaled percent]
   [-min voltage]
   [-max voltage]
   [-rail_reference voltage]
   [-states state_list]
```

Data Types

```
   name             string
   percent          float
   voltage          float
   state_list       list
```

ARGUMENTS

-model *name*

The name of the SPICE transistor model affected by the command options.

-conservative

Causes NanoTime Ultra to set the upper and lower bounds on the body voltage for each transistor model.

-average

Causes NanoTime Ultra to set the body voltage for each transistor model to a single typical value between the "conservative" upper and lower bounds.

-scaled *percent*

Causes NanoTime Ultra to set the upper and lower bounds on the body voltage for each transistor

model, using values that are scaled between the conservative and average methods. The scaling factor is specified as a percentage, from 0 to 100. A scaling of 50 results in values halfway between the conservative and average methods. A scaling of 0 is the same as the conservative method and a scaling of 100 is the same as the average method.

-min ***voltage***

Specifies the lower limit on the body voltage. This option must be used with the **-max** and **-rail_reference** options.

-max ***voltage***

Specifies the upper limit on the body voltage.

-rail_reference ***voltage***

Specifies the rail (power supply) voltage. If multiple supply voltages exist in the design, the minimum and maximum boundary values are adjusted according to the difference between the applied voltage and the rail reference voltage.

-states ***state_list***

The initial gate-source-drain states for which the command applies: 0 for the "off" state, 1 for the "on" state, or r for "tied-to-rail" state. The elements of the list can be 0r0, 0r1, 1r0, 000, 001, 100, 010, 011, and 111.

DESCRIPTION

The **set_soi_parameters** command provides the information necessary for NanoTime Ultra to determine the initial body voltage of silicon-on-insulator (SOI) transistors. The command applies to all transistors of a given SPICE model or to all models. You cannot set this information on individual instances in the design. However, you can specify body contact information on instances with the **set_soi_transistor_type** command.

You must have a NanoTime Ultra license to analyze SOI transistors.

You can let the tool estimate the body voltage ranges by using the **-conservative**, **-average**, or **-scaled** options. Alternatively, you can specify the possible ranges explicitly with the **-min**, **-max**, and **-rail_reference** options.

The **-conservative** option uses worst-case estimates for the body voltage ranges for each gate-source-drain state. The timing results are likely to be pessimistic, but detection of timing violations due to the floating body effect is ensured.

The **-average** option uses a typical estimated body voltage for each gate-source-drain state, using the same value for both the minimum and maximum. The timing results are likely to be optimistic.

The **-scaled** option uses body voltage values that are scaled between the conservative and average methods by a specified factor between 0 and 100 percent. A scaling factor of 50 results in values halfway between the conservative and average methods. A scaling factor of 0 is the same as the conservative method and a scaling factor of 100 is the same as the average method.

A transistor that changes state infrequently is likely to have a more extreme body voltage, so using the conservative method or a scaling factor closer to 0 percent is appropriate in this case. A transistor that changes state frequently is likely to have a moderate, intermediate body voltage, so using an intermediate scaling factor such as 50 to 70 percent is appropriate in this case.

Instead of allowing NanoTime Ultra to determine the body voltage range, you can explicitly specify the upper and lower bounds by using the **-max**, **-min**, and **-rail_reference** options.

The **-states** option specifies the initial gate-source-drain states for which the command applies: 0 for the "off" state, 1 for the "on" state, or r for "tied-to-rail" state. The elements of the list can contain any of the following: 0r0, 0r1, 1r0, 000, 001, 100, 010, 011, and 111. For an NMOS transistor, 0 means a low voltage, 1 means a high voltage, and r means the source tied to ground. For a PMOS transistor, 0 means a high voltage, 1 means a low voltage, and r means the source tied to Vdd.

The states 101 and 110 are not allowed because they represent the transistor in an unstable state, with the transistor turned on and the source and drain at opposite voltages.

To report the SOI settings, use the **report_technology -soi_parameters** command.

COMMAND PHASING

The **set_transition_coefficients** command should be executed between the "topology-checked" and "design-checked" states (in other words, after the **check_topology** command and before the **trace_path** or **extract_model** commands).

EXAMPLES

The following command invokes the "conservative" method to determine body voltage ranges for all transistors.

```
nt_shell> set_soi_parameters -conservative
```

The following command sets SOI parameters for model "nch" that are applicable to transistors in the 0r0, 000 and 010 states.

```
nt_shell> set_soi_parameters -model nch \
          -min 0.2 -max 0.8 -rail_reference 1.8 \
          -states {0r0 000 010}
```

SEE ALSO

```
report_technology(2)
set_soi_transistor_type(2)
write_spice(2)
soi_enable_analysis(3)
```

set_soi_transistor_type

Specifies a body contact type for a list of transistor instances.

SYNTAX

```
string set_soi_transistor_type
   [-body2source]
   [-body_contact voltage]
   cell_list
```

Data Types

```
   voltage         float
   cell_list       list
```

ARGUMENTS

-body2source

Specifies that the listed transistor instances have the body and source terminals connected together.

-body_contact *voltage*

Specifies that the listed transistor instances have the body connected to a voltage source at the specified voltage.

cell_list

List or collection of transistor cells.

DESCRIPTION

The **set_soi_transistor_type** command assigns a body contact type to a specified list of silicon-on-insulator (SOI) transistors in the design. The body connection type affects the results of simulation and the resulting delay and slew of transitions.

You must have a NanoTime Ultra license to analyze SOI transistors.

The **-body2source** option specifies that the listed transistors have the body and source terminals connected together. The **-body_contact** option specifies that the listed transistors have the body connected to a constant voltage source at the specified voltage.

If you do not specify a body connection type for an SOI transistor, the body is assumed to be floating. The upper and lower bounds on the voltage of a floating body can be set for a transistor model with the **set_soi_parameters** command.

COMMAND PHASING

The **set_transition_coefficients** command should be executed between the "topology-checked" and "design-checked" states (in other words, after the **check_topology** command and before the **trace_path** or **extract_model** commands).

EXAMPLES

The following command sets SOI transistor type for a transistor x00.mp1 to a body contact voltage of 1.2 volts.

```
nt_shell> set_soi_transistor_type -body_contact 1.2 x00.mp1
```

SEE ALSO

```
set_soi_parameters(2)
write_spice(2)
soi_enable_analysis(3)
```

set_steady_state_resistance

Specifies the steady-state holding resistance value to use for injection noise calculation on nets driven by specified model library pins.

SYNTAX

```
status set_steady_state_resistance
   [-above]
   [-below]
   [-low]
   [-high]
   resistance_value
   lib_pin_list
```

Data Types

resistance_value	float
lib_pin_list	list

ARGUMENTS

-above

Specifies steady-state resistance for noise above the ground or power rail.

-below

Specifies steady-state resistance for noise below the ground or power rail.

-low

Specifies steady-state resistance for ground rail noise.

-high

Specifies steady-state resistance for power rail noise.

resistance_value

The steady-state resistance value in ohms. The value must be greater than or equal to 0.01ohms.

lib_pin_list

A list of model library pins in the current design on which to set the steady-state resistance value.

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. Use the **set_steady_state_resistance** command to set the resistance value, in ohms, for the specified model library pins.

You can use combinations of the **-above**, **-below**, **-low**, and **-high** options to specify the resistance values for different types of noise.

The **set_steady_state_resistance** command sets resistance values on specific library pins. To provide default values to use for all other library pins, use the **lib_pin_steady_state_resistance_above_high** variable and the similarly named variables for other types of noise.

COMMAND PHASING

The **set_steady_state_resistance** command can be executed at any time after the **link_design** command.

SEE ALSO

```
update_noise(2)
remove_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_high(3)
```

set_supply_net

Identifies power supply or ground nets in the design.

SYNTAX

```
status set_supply_net
   [-gnd]
   [-virtual]
   net_list
```

Data Types

```
   net_list      list
```

ARGUMENTS

-gnd

Specifies ground nets. In the absence of this option, the command specifies supply nets.

-virtual

Specifies a virtual supply, representing a transistor connection to a true voltage rail.

net_list

The list of nets that are ground nets or Vdd nets.

DESCRIPTION

This command can be used to identify power supply or ground nets in the design when this information is not available from the netlist.

To specify nets in the design that are power supply nets, use the **set_supply_net** command without the **-gnd** option and provide a list of nets. NanoTime marks those nets by setting the net attributes **is_supply**

and **is_vdd** to **true**.

To specify nets in the design that are ground nets, use the **set_supply_net** command with the **-gnd** option and provide a list of nets. NanoTime marks those nets by setting the net attributes **is_supply** and **is_ground** to **true**.

To specify nets in the design that are virtual supply nets, use the **-virtual** option and provide a list of nets. NanoTime marks those nets by setting the net attribute **is_virtual_supply** to **true**. The tool also sets the **is_virtual_vdd** or **is_virtual_ground** attribute appropriately, depending on whether you use the **-gnd** option. A NanoTime Ultra license is required to define a virtual supply net.

A virtual supply is used like an actual power supply net. Typically, it is connected to a true supply or ground net through a single transistor (sometimes called a sleeper transistor) or multiple transistors having a very low resistance. NanoTime automatically recognizes the transistors between the virtual supply and the true supply as a power switch topology. A net should not be defined as both a true supply and a virtual supply.

By default, the virtual supply voltage is set to the same value as the true supply voltage. Use the **set_voltage** command to set the virtual supply voltage if it is different from the true supply voltage.

The automatic transistor direction setting algorithm in NanoTime uses the virtual supply definitions to more accurately and completely determine the actual direction of transistor operation.

To cancel the effects of the **set_supply_net** command, use the **remove_supply_net** command.

To have NanoTime automatically identify power and ground nets based on the net naming conventions, set the variables **link_vdd_alias** and **link_gnd_alias** before executing the **link_design** command.

COMMAND PHASING

The **set_supply_net** command is typically executed in the "netlist-linked" state (in other words, after the **link_design** command and before a **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 **set_supply_net** command appears after the first **match_topology** command, the **set_supply_net** command 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.

Commands that appear after the **set topo_match_topology_reset_clock_and_topology true** command 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.

If you use the **set_supply_net** command after the **check_topology** command, the NanoTime tool transitions back to the "netlist-linked" state and discards all operations performed after the **check_topology** command.

If you use the **set_supply_net** command in a topology object created with the **create_lib_topology** command, specify the **-action_pre_match_topology** option.

EXAMPLES

The following command identifies all the nets whose names match "g1*" as ground nets.

```
nt_shell> set_supply_net -gnd g1*
```

SEE ALSO

```
create_lib_topology(2)
remove_supply_net(2)
set_voltage(2)
link_gnd_alias(3)
link_vdd_alias(3)
topo_match_topology_reset_clock_and_topology(3)
```

set_technology

Specifies the transistor technologies to use for simulation and analysis of the current design.

SYNTAX

```
status set_technology
   [-min min_name]
   [-max max_name]
   [-min_clock min_clock_name]
   [-max_clock max_clock_name]
   [name]
```

Data Types

```
   min_name            string
   max_name            string
   min_clock_name      string
   max_clock_name      string
   name                string
```

ARGUMENTS

-min *min_name*

Specifies the name of the technology used to calculate minimum (shortest-path) delays. If used, the **-max** option must also be used.

-max *max_name*

Specifies the name of the technology used to calculate maximum (longest-path) delays. If used, the **-min** option must also be used.

-min_clock *min_clock_name*

Specifies the name of the technology used to calculate maximum delays for clock paths, if different from the **-min** setting. If used, the **-max_clock** option must also be used.

-max_clock *max_clock_name*

Specifies the name of the technology used to calculate maximum delays for clock paths, if different

from the **-max** setting. If used, the **-min_clock** option must also be used.

name

Specifies the name of the single technology used to calculate all delays.

DESCRIPTION

The **set_technology** command specifies the transistor technologies to use for simulation and analysis of the current design.

A technology contains transistor models that specify the electrical parameters of transistors under specified conditions of temperature, voltage, and process. You create a technology in NanoTime by reading a techfile with the **read_techfile** command, by reading a SPICE model directly with the **read_spice_model** command, or by reading a SPICE model indirectly using a .lib statement in a SPICE netlist file.

The design has four separate technology settings:

- min -- for minimum data paths
- max -- for maximum data paths
- min_clock -- for minimum clock paths
- max_clock -- for maximum clock paths

For a setup check, NanoTime uses maximum delays and longest paths for the launch clock path and data path, and minimum delays and shortest paths for the capture clock path.

For a hold check, NanoTime uses minimum delays and shortest paths for the launch clock path and data path, and maximum delays and longest paths for the capture clock path.

The **set_technology** command specifies which technologies (transistor models) to use for which paths. In the command, you must specify the technologies for the paths in one of the following three ways:

1. A single technology, such as:

nt_shell> **set_technology TYP**

2. min and max technologies, such as:

nt_shell> **set_technology -min MINp1 -max MAXp1**

3. min, max, min_clock, and max_clock technologies, such as:

nt_shell> **set_technology -min MINp1 -max MAXp1 \ -min_clock MAXp1 -max MINp1**

If you specify a single technology, it is assigned to all four path types like this:

```
min technology = TYP
max technology = TYP
min_clock technology = TYP
max_clock technology = TYP
```

In that case, NanoTime uses the same technology for all analysis.

If you specify a min technology and a max technology, the two technologies are assigned to the four path types like this:

```
min technology = MINp1
max technology = MAXp1
min_clock technology = MINp1
max_clock technology = MAXp1
```

In that case, NanoTime uses the worst-case technology for each path segment in a timing check. This is the most conservative kind of checking.

You can also specify a technology for each of the four path types using **-min**, **-max**, **-min_clock**, and **-max_clock**. In the example above, the resulting settings would be:

```
min technology = MINp1
max technology = MAXp1
min_clock technology = MAXp1
max_clock technology = MINp1
```

In this example, NanoTime uses the technology MINp1 for all hold checking, and the technology MAXp1 for all setup checking.

When you execute the **set_technology** command, NanoTime links each transistor in the design to a transistor model and reports the number of transistors linked, as in the following example:

```
Linking transistor models...
Information: 11032 out of 11032 transistors have
transistor models linked to them.  (TECH-003)
1
```

If not all the transistors in the design are linked to models, you get a warning message similar to the following:

```
Linking transistor models...
Warning: 478 out of 11032 transistors do not have
transistor models linked to them.  (TECH-002)
1
```

This can happen when you are using techfile-based technologies, and there are no available models to match some of the transistors in the design. To increase the usage of techfile-based models, you can increase the allowable difference between transistor parameter values in the design and the parameter values of the models. To do this, set the following variables to nonzero values:

```
tech_match_length_pct
tech_match_width_pct
tech_match_param_pct
```

Before you can analyze the design, each transistor must have a linked model. The **check_design** command verifies that this requirement is satisfied.

To get of list of technologies that are available in NanoTime, use the **list_technology** command.

To view the current settings for the design made with the **set_technology** command, use the

report_design command. For example:

```
nt_shell> report_design
 ...
Technology:
  min_technology              MAXp1
  max_technology              MINp1
  min_clock_technology        MINp1
  max_clock_technology        MAXp1
 ...
```

To get a report on the model characteristics of a technology, use the **report_technology** command.

COMMAND PHASING

The **set_technology** command must be executed after the **link_design** command and before the **check_design** command.

EXAMPLES

The following session reads in two techfile-based technologies, reads and links a design, and specifies the technologies to use for min and max analysis.

```
nt_shell> read_techfile -name fast /techdir/tech1.gz
1
nt_shell> read_techfile -name slow /techdir/tech2.gz
1
nt_shell> list_technology
Technology Registry

Attributes (T):
    t - Tech data from techfile
    m - Tech data from SPICE model
    s - SPICE model

Name      Voltage    Temp Nmos    Pmos    T Source file
--------- ------- ------- ------- ------- - -------------
slow        1.250 110.000 n       p       t tech2.gz
fast        1.200 110.000 n       p       t tech1.gz
1
nt_shell> register_netlist mydesign.sp
1
nt_shell> link_design top
 ...
Linking design top...
Design 'top' was successfully linked.
1
nt_shell> set tech_match_length_pct 10
10
nt_shell> set tech_match_param_pct 10
10
nt_shell> set tech_match_width_pct 10
10
nt_shell> set_technology -min fast -max slow
```

```
Linking transistor models...
Information: 3256 out of 3256 transistors have all
transistor models linked to them.  (TECH-003)
1
nt_shell> report_technology
Model  Tech.
Name   Name   Volt  Temp   Width  Length  Delvto  Mulu0  Total
------ ------ ----- ------ ------ ------- ------- ------ -----
n      slow   1.25  110.00 70     19      0.000   1.000  376
p      slow   1.25  110.00 70     19      0.000   1.000  376
p      fast   1.20  110.00 384    8       0.000   1.000  425
n      fast   1.20  110.00 326    20      0.000   1.000  533
1
nt_shell> report_design
  ...
Technology:
  max_technology                slow
  min_technology                fast
  max_clock_technology          slow
  min_clock_technology          fast
  ...
```

SEE ALSO

```
list_technology(2)
read_techfile(2)
read_spice_model(2)
report_design(2)
report_technology(2)
link_enable_netlist_spice_model_linking(3)
tech_default_voltage(3)
tech_match_length_pct(3)
tech_match_width_pct(3)
tech_match_param_pct(3)
tech_netlist_spice_model_name(3)
```

set_timing_check_attachment

Sets the timing check attachment point for a topology.

SYNTAX

```
status set_timing_check_attachment
   [-setup_to latch_net | input | output]
   [-hold_to  latch_net | input | evaluate_net]
   [-quiet]
   topo_objects
```

Data Types

```
   topo_objects       list
```

ARGUMENTS

-setup_to

Specifies setup type of the topologies. (Values: input, output, latch_net)

-hold_to

Specifies hold type of the topologies. (Values: input, latch_net, evaluate_net)

-quiet

Suppresses messages from this command.

topo_objects

List of topology objects for changing the timing check attachment.

DESCRIPTION

The **set_timing_check_attachment** command changes the timing check reference point of specified

topologies in the design. Valid topologies are latches, flip-flops, and precharge structures.

The **-setup_to** option applies to latches, flip-flops, and precharge structures. Valid setup types are **input**, **output** and **latch_net**.

The **-hold_to** option applies to latches, flip-flops, and precharge structures. Valid hold types for latches and flip-flops are **input** and **latch_net**. Valid hold types for precharge structures are **input** and **evaluate_net**.

COMMAND PHASING

The **set_timing_check_attachment** command is typically executed in the "topology-checked" state (between the **check_topology** command and the **check_design** command). If you execute it later, NanoTime transitions back to the "topology-checked" state and discards all operations performed after the **check_design** command.

EXAMPLES

The following command causes the setup time of the latch at N1 to be calculated using the delay to the latch output.

```
nt_shell> set_timing_check_attachment -setup_to output \
          [get_topology -of_object N1 -structure_type latch]
```

SEE ALSO

topo_latch_setup_to(3)
topo_latch_hold_to(3)

set_timing_check_attributes

Modifies the setup or hold time of an existing timing check.

SYNTAX

status **set_timing_check_attributes**

Options That Select a Timing Check to Modify

[-from *from_object* | -rise_from *from_object* | -fall_from *from_object*] -to *to_object* | -rise_to *to_object* | -fall_to *to_object* [-target *target_object* | -rise_target *target_object* | -fall_target *target_object*] [-trigger *trigger_object* | -rise_trigger *trigger_object* | -fall_trigger *trigger_object*] [-to_not_target] [-of_objects *objects*]

Options That Modify a Timing Check

[-label *name*] [-setup | -hold] [-continue | -continue_with_error_adjustment | -stop] [-transparent] [-switch_type] [-zero_positive_slack] [-always_pbsa] [-selective_pbsa] [-force_same_phase] [-force_no_same_phase] [-force_max_max] [-force_min_min] [-quiet] [-margin] [-rise_from_transition | -fall_from_transition] [-rise_target_transition | -fall_target_transition] *check_value*

Data Types

from_object	list
to_object	list
target_object	list
trigger_object	list
objects	list
name	string
check_value	list

ARGUMENTS

-from *from_object*

Specifies the clock or reference port or pin of the timing check.

-rise_from *from_object*

Similar to the **-from** option, but applies only to rising transitions at the clock pin.

-fall_from ***from_object***

Similar to the **-from** option, but applies only to falling transitions at the clock pin.

-to ***to_object***

Specifies the data pin or port in the current design where the timing check applies. This is the checked pin of the timing check.

-rise_to ***to_object***

Similar to the **-to** option, but applies only to rising signals at the data pin.

-fall_to ***to_object***

Similar to the **-to** option, but applies only to falling signals at the data pin.

-target ***target_object***

The target port or pin at the output of the simulation unit related to the timing check, where path tracing ends, which might be the same as the timing check point ("to" object).

-rise_target ***target_object***

Similar to the **-target** option, but applies only to rising signals at the target object.

-fall_target ***target_object***

Similar to the **-target** option, but applies only to falling signals at the target object.

-trigger ***trigger_object***

The trigger port or pin at the input of the simulation unit related to the timing check, where path tracing starts.

-rise_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to rising signals at the trigger object.

-fall_trigger ***trigger_object***

Similar to the **-trigger** option, but applies only to falling signals at the trigger object.

-to_not_target

Specifies that the "to" object is different from the "target" object. In the absence of this option, a "to" object specified by itself is considered to be both the timing check point and the "target" object at the output of the simulation unit, where path tracing ends.

-of_objects ***objects***

Modifies the timing checks of the objects in a collection previously created by a **get_timing_checks** command.

-label ***name***

Specifies a new label for the check.

-setup

Specifies that the check value applies to setup checking only.

-hold

Specifies that the check value applies to hold checking only. If neither of the **-setup** or **-hold** options is present, the check value applies to both setup and hold checks.

-continue

Continue paths tracing through the timing endpoint after evaluation of the timing check.

-continue_with_error_adjustment

Adjusts the arrival, if necessary, and continues path tracing through the timing endpoint after evaluation of the timing check. In the case of a setup check, the adjustment pulls in the arrival time to make the error zero. In the case of a hold check, the adjustment pushes out the arrival time to make the error zero.

-stop

Stops path tracing at the timing endpoint after evaluation of the timing check. This cancels the **-continue** option used in a **create_timing_check** command.

-transparent

Specifies that the check is performed at the close of transparency.

-switch_type

Converts a data-to-data check from setup to hold or vice versa.

-zero_positive_slack

Adjusts the path delay for a timing check (other than data-to-data checks) to yield zero slack under worst case.

-always_pbsa

Applies path-based slack adjustment unconditionally.

-selective_pbsa

Applies path-based slack adjustment if the adjustment value exceeds the threshold specified by the **pbsa_min_threshold** variable.

-force_same_phase

Forces this timing check to be evaluated using the same cycle between the capture and launch clocks. This overrides topology-related phasing controls but not path-related ones.

-force_no_same_phase

Forces this timing check to be evaluated using the next cycle between the capture and launch clocks. This overrides topology-related phasing controls but not path-related ones.

-force_max_max

Forces this timing check to be evaluated using max reference arrival and max checked arrival. This overrides all other path length controls. It applies to all timing checks. It is recommended that the label of the check be changed for better clarity in path reporting.

-force_min_min

Forces this timing check to be evaluated using min reference arrival and min checked arrival. This overrides all other path length controls. It applies to all timing checks. It is recommended that the label of the check be changed for better clarity in path reporting.

-quiet

Suppresses warning messages.

-margin

Defines the specified *check_value* to be a timing margin that is applied to the existing setup or hold time. A positive margin value results in a more restrictive timing check and lower reported slack. Without this option, the specified *check_value* is the new setup or hold time requirement for the timing check. This option cannot be used with a timing check lookup table.

-rise_from_transition

Specifies that the first value of each triplet in the check value lookup table is the rail-to-rail transition time of a rising signal at the "from" pin of the timing check.

-fall_from_transition

Specifies that the first value of each triplet in the check value lookup table is the rail-to-rail transition time of a falling signal at the "from" pin of the timing check.

-rise_target_transition

Specifies that the second value of each triplet in the check value lookup table is the rail-to-rail transition time of a rising signal at the "target" pin of the timing check.

-fall_target_transition

Specifies that the second value of each triplet in the check value lookup table is the rail-to-rail transition time of a falling signal at the "target" pin of the timing check.

check_value

If the **-margin** option is not used, the *check_value* is the timing check requirement. If the **-margin** option is used, the *check_value* is a timing margin that is applied to the existing setup or hold time requirement. A positive value results in a more restrictive timing check and smaller (more negative) reported slack. A negative value results in a less restrictive check and a larger (more positive) reported slack.

The *check_value* can also be a list of triplets for a setup or hold time lookup table.

DESCRIPTION

Use the **set_timing_check_attributes** command to modify any existing timing check, whether NanoTime created it automatically or you created it manually. You must modify a timing check after the **check_design** command, but before the **trace_paths** or **extract_model** command.

The **set_timing_check_attributes** command is an object-based timing exception command. Object-based timing exception commands have lower priority than path-based timing exceptions specified with the **set_same_phase_path**, **set_no_same_phase_path**, **set_multicycle_path**, **set_false_path**, or **set_no_timing_check_path** commands.

The options for the **set_timing_check_attributes** command are of two types: those that specify which timing checks to modify, and those that specify the changes to apply.

Timing Check Selection

For more information about timing checks, see the NanoTime user guide.

A timing check might be defined by up to four pins, as follows:

```
-- "from" object: clock or reference pin of the timing check
-- "to" object: data pin of the timing check
-- "trigger" object: input pin of the simulation unit
-- "target" object: output pin of the simulation unit
```

NanoTime performs timing checks by considering the arrival times at the specified "from" and "to" points. The "from" object is a port or pin where there is a clock signal, or the related pin of a data-to-data timing check (also known as the reference pin). The "to" object is a port or pin where there the timing check is evaluated (also known as the checked pin or constrained pin). The timing check might optionally have a "trigger" pin and a "target" pin, which are the startpoint and endpoint of a delay arc that must be traversed for the timing check to be evaluated.

In other words, the "from" and "to" points specify where the timing checks are performed, while the "trigger" and "target" points specify where path tracing is performed. To narrowly define the timing checks affected by the **set_timing_check_attributes** command, you can specify all four objects. Specifying fewer objects affects more timing checks. For example, if you specify only the "from" object, the command affects all timing checks where the "from" object is the clock or timing reference pin. Those timing checks might have different "to," "trigger," and "target" objects.

By default, NanoTime assumes that a "to" object specified by itself is also the "target" object, so it looks for timing checks where the "to" object is the constrained pin of the timing check and is also the endpoint of path tracing. This type of timing check is associated with a pin, not a timing arc.

If the timing check is associated with a timing arc rather than a pin, the "target" object is not the same as the "to" object. To look for this type of timing check, either specify the "target" object explicitly or use the **-to_not_target** option. NanoTime looks for timing arcs where the timing check is performed at the "to" object, but path tracing stops at the "target" object.

In a master-slave flip-flop structure, the trigger object is the latch node of the master latch and the target is the latch node of the slave latch. The simulation unit consists of the circuitry from the trigger to the target. However, unlike a simple latch, the flip-flop captures data on the opening edge rather than the closing edge, and the target pin is nontransparent rather than transparent. The "from" object is the clock pin of the slave latch, and the "to" object (where the data arrival time is measured) is the data input pin of the slave latch.

If you are modifying a timing check created earlier by the **create_timing_check** command, the "from" and "to" objects in the **set_timing_check_attributes** command must match the ones used in the original **create_timing_check** command, as well as the **-setup** and **-hold** options, if used originally.

Timing Check Lookup Tables

Instead of a fixed check value, you can specify a lookup table in which the check value is a function of the transition time of the signal at the "from" pin or at the "target" pin, or both. To do so, specify the check value as list of triplets. For example:

```
nt_shell> set tchk [get_timing_checks \
          -from ... -to ... ]
 ...
nt_shell> set_timing_check_attributes \
          -of_objects $tchk \
          -fall_from_transition -rise_target_transition \
          { {1.0 2.0 2.9} {1.0 3.2 4.4} {1.0 5.0 6.0} \
          {2.5 2.0 5.1} {2.5 3.2 6.5} {2.5 5.0 7.7} \
          {4.0 2.0 7.0} {4.0 3.2 7.4} {4.0 5.0 8.3} }
```

The lookup table in this example has nine triplets, each containing three floating-point time values.

Within each triplet, the first number is the rail-to-rail transition time at the reference ("from") pin of the timing check. The second number is the rail-to-rail transition time at the pin where the path tracer has reached for an in-trace constraint, or where the constraint is evaluated for a data-to-data constraint. The third entry is the check value corresponding to the two transition time values.

In the example, the nine triplets specify a lookup table like this:

```
"from" trans.  "target" trans.  check value
     1.0            2.0            2.9
     1.0            3.2            4.4
     1.0            5.0            6.0

     2.5            2.0            5.1
     2.5            3.2            6.5
     2.5            5.0            7.7

     4.0            2.0            7.0
     4.0            3.2            7.4
     4.0            5.0            8.3
```

The check value is a function of the two transition times: the falling transition time at the "from" pin and the rising transition time at the "target" pin. To determine the exact check value, NanoTime uses interpolation between the transition times specified in the table, or extrapolation of the table for transition times outside the specified range.

If you use the **-rise_from** option with a lookup table, the first number in each triplet represents the rising transition time at the "from" object. If you use the **-from** option, the first number in each triplet applies to both rising and falling transitions at the "from" object. In the latter case, you can optionally use the **-rise_from_transition** or **-fall_from_transition** option to restrict the edge type represented by the first number in each triplet, either rising or falling.

In a similar manner, with the **-target** option, you can use the **-rise_target_ transition** or **-fall_target_transition** option to specify the type of edge represented by the second number in each triplet, either rising or falling edges at the "target" pin.

The foregoing nine-triplet example is a two-dimensional table, in which the check value is a function of two variables. You can also make a one-dimensional table that depends only one transition time, either the "from" transition or the "target" transition. For example:

```
nt_shell> set tchk [get_timing_checks \
          -from ... -to ... ]
 ...
```

```
nt_shell> set_timing_check_attributes \
          -of_objects $tchk \
          -fall_from_transition -rise_target_transition \
          { {1.0 2.0 2.9} {1.0 3.2 4.4} {1.0 5.0 6.0}
```

In this example, only the transition times at the "target" pin are varied.

For a two-dimensional table, there must be at least four triplets, but preferably nine or more for good interpolation and extrapolation accuracy. For a one-dimensional table, there must be at least two triplets, but preferably three or more for good accuracy.

COMMAND PHASING

The **set_timing_check_attributes** command must be executed after the **check_design** command but before the **trace_paths** command.

EXAMPLES

The following commands assign new setup and hold times to existing timing checks. The "from" and "to" specifications must exactly match the ones used in the original **create_timing_check** commands.

```
nt_shell> set_timing_check_attributes \
          -rise_from [get_ports CLK] \
          -to [get_ports DATA1] -setup 0.4

nt_shell> set_timing_check_attributes \
          -rise_from [get_ports CLK] \
          -to [get_ports DATA1] -hold 0.6
```

The following command sets a margin of 0.1 time unit for both the setup and hold checks. This causes the reported slack to be reduced (made more negative) by 0.1 time unit.

```
nt_shell> set_timing_check_attributes \
          -rise_from [get_ports CLK] \
          -to [get_ports DATA1] -margin 0.1
```

SEE ALSO

```
create_timing_check(2)
set_data_check(2)
get_timing_checks(2)
remove_timing_check(2)
remove_data_check(2)
trace_paths(2)
pbsa_min_threshold(3)
timing_default_phasing(3)
set_phasing(2)
set_same_phase_path(2)
```

```
set_no_same_phase_path(2)
```

set_transistor_direction

Sets the signal flow direction of a specified transistor.

SYNTAX

```
status set_transistor_direction
   -transistor value
     | -to_net to_nets
     | -from_net from_nets
   [cell_list]
```

Data Types

```
value        string
to_nets      list
from_nets    list
cell_list    list
```

ARGUMENTS

-transistor *value*

Sets the signal flow direction of the transistors listed in the *cell_list* to one of the following: **s2d** (source-to-drain), **d2s** (drain-to-source), **bidi** (bidirectional), or **none** (nonconducting).

-to_net

Sets the signal flow direction to be toward the specified net for all transistors connected to the net or for the transistors listed in the *cell_list*.

-from_net

Sets the signal flow direction to be away from the specified net for all transistors connected to the net or for the transistors listed in the *cell_list*.

cell_list

Specifies the transistors for which the signal flow direction is marked.

DESCRIPTION

The **set_transistor_direction** command explicitly specifies the signal flow direction of one or more transistors. NanoTime marks this information on the specified transistors. During topology recognition and path tracing, NanoTime uses this information to determine how the transistor operates.

You must specify the signal flow direction using one of the following options:

```
-transistor: direction based on transistor terminals
-to_net: direction toward a specified net
-from_net: direction away from a specified net
```

Using the **-transistor** option, you specify the direction as **s2d**, **d2s**, **bidi**, or **none**, meaning source-to-drain, drain-to-source, bidirectional, or nonconducting, respectively. You must specify the transistor cell names in the command. You can use the **report_cell** command to get a list of cell names in the current design and current instance.

If you use the **-to_net** or **-from_net** option, the command sets the signal flow direction toward or away from the specified net or nets, for all transistors listed in the command. If no transistors are listed in the command, it sets the direction for all transistors connected to the specified nets.

If specified after the **match_topology** command and the variable **topo_match_topology_reset_clock_and_topology** is **true**, NanoTime repropagates the clock network and rerecognizes topologies in the next **match_topology** or **check_topology** command. If specified in lib_topologies, it must be specified within the pre_match_topology phase point.

To get a report on the number of transistors that are unidirectional and bidirectional, use the **report_transistor_direction** command.

COMMAND PHASING

The **set_transistor_direction** command can only be executed in the "netlist-linked" state (in other words, after the **link_design** command and before the **check_topology** command). If you use the command later in the flow, the NanoTime tool transitions back to the "netlist-linked" state and discards all commands and operations performed in the later phases.

EXAMPLES

The following example reports the transistor names in cell Xxor7.X1 and then sets the transistor direction to source-to-drain for transistor Mn0.

```
nt_shell> report_cell Xxor7.X1.*
 ...
Attributes:
    t - transistor
    c - capacitor
    r - resistor
    h - hierarchical
```

```
Cell            Reference       Library    Attributes
--------------- --------------- ---------- ----------
Mp0             transistor                 t
Mn0             transistor                 t
--------------------------------------------------
Total 2 cells

1
nt_shell> set_transistor_direction -transistor s2d  \
          [get_cell Xxor7.X1.Mn0]
1
```

The following command applies the **set_transistor_direction** command to all instances of the subcircuit MUX, thereby specifying the signal direction as source-to-drain for transistor MN1 in MUX.

```
nt_shell> foreach_match MUX
          -command { set_transistor_direction \
          -transistor s2d MN1 }
```

SEE ALSO

create_lib_topology(2)
current_design(2)
current_instance(2)
match_topology(2)
report_cell(2)
report_transistor_direction(2)
set_transistor_drive_factor(2)
set_transistor_parameter(2)
topo_match_topology_reset_clock_and_topology(3)

set_transistor_drive_factor

Modifies the drive strength of specified transistors.

SYNTAX

```
status set_transistor_drive_factor
   [-min]
   [-max]
   [-rise]
   [-fall]
   factor
   cell_list
```

Data Types

```
   factor       float
   cell_list    list
```

ARGUMENTS

-min

Sets the drive factor only for minimum (short path) analysis.

-max

Sets the drive factor only for maximum (long path) analysis.

-rise

Sets the drive factor only for a rising signal at the trigger pin.

-fall

Sets the drive factor only for a falling signal at the trigger pin.

factor

The drive factor, a positive floating-point number, which modifies the drive strength of the specified transistors.

cell_list

The list of transistors affected by the command.

DESCRIPTION

This command changes the drive strength of transistors in the design by a specified factor. For example, specifying a factor of 2.0 doubles the drive strength of the specified transistors, like increasing the transistor width-to-length ratio by a factor of 2. In the command, you must specify the drive strength factor and the list of transistors in the design that are to be changed.

You can also use the **-min** or **-max** option to restrict the drive adjustment factor to just minimum (short path) or just maximum (long path) analysis; and the **-rise** or **-fall** option to restrict the drive adjustment factor to just rising or just falling edges at the trigger pin of the transistor. In the absence of these options, the factor applies to all conditions.

To cancel a drive factor that has been applied to a transistor, use the **remove_transistor_drive_factor** command.

COMMAND PHASING

The **set_transistor_drive_factor** command can only be executed between "netlist-linked" and "design-checked" states. If used after "design-checked", NanoTime transitions back to the "design-checked" state and discards all commands and operations performed in the later phase(s).

EXAMPLES

The following example adjusts the drive strength of transistor X3.Mp0 by a factor of 1.2 for minimum (short path) analysis and by a factor of 0.8 for maximum (long path) analysis.

```
nt_shell> set_transistor_drive_factor \
          -min 1.2 [get_cells X3.Mp0]

nt_shell> set_transistor_drive_factor \
          -max 0.8 [get_cells X3.Mp0]
```

SEE ALSO

```
remove_transistor_drive_factor(2)
report_attribute(2)
set_transistor_direction(2)
set_transistor_parameter(2)
```

set_transistor_parameter

Sets parameters of specified transistors.

SYNTAX

```
status set_transistor_parameter
   [-model model_name]
   [-width value]
   [-length value]
   [-ad value]
   [-as value]
   [-delvto value]

   [-sa1 value]
   [-sa2 value]
   [-sa3 value]
   [-sa4 value]
   [-sa5 value]
   [-sa6 value]
   [-sa7 value]
   [-sa8 value]
   [-sa9 value]
   [-sa10 value]

   [-sb1 value]
   [-sb2 value]
   [-sb3 value]
   [-sb4 value]
   [-sb5 value]
   [-sb6 value]
   [-sb7 value]
   [-sb8 value]
   [-sb9 value]
   [-sb10 value]

   [-sw1 value]
   [-sw2 value]
   [-sw3 value]
   [-sw4 value]
   [-sw5 value]
   [-sw6 value]
   [-sw7 value]
   [-sw8 value]
   [-sw9 value]
   [-sw10 value]

   [-sca value]
   [-scb value]
   [-scc value]
   [-sc value]
```

```
[-mulvsat value]
[-factuo value]
[-mulu0 value]
[-mulid0 value]

[-nfin value]
[-tfin value]
[-d value]
[-fpitch value]
[-aseo value]
[-adeo value]
[-pseo value]
[-pdeo value]
[-asej value]
[-adej value]
[-psej value]
[-pdej value]
[-lrsd value]
[-xl value]
[-delvtrand value]
[-u0mult value]
[-ids0mult value]
[-cgsp value]
[-cgdp value]
[-cdsp value]
[-ngcon value]

[-nrd value]
[-nrs value]
[-pd value]
[-ps value]
[-bvs_gain value]
[-name parameter_name]
[-value value]
[-quiet]

object_list
```

Data Types

```
model_name        string
value             float
parameter_name    string
object_list       collection
```

ARGUMENTS

-model *model_name*

The name of the transistor model.

-width *value*

The width of the transistor channel in microns.

-length *value*

The length of the transistor channel in microns.

-ad *value*

The area of the drain in square microns.

-as *value*

The area of the source in square microns.

-delvto *value*

Delta Vt offset, a shift in the transistor threshold voltage. The default is 0.0.

-sa1 *value*

LOD (length of diffusion) SA parameter in microns.

-sa2 *value*

LOD (length of diffusion) SA parameter in microns.

-sa3 *value*

LOD (length of diffusion) SA parameter in microns.

-sa4 *value*

LOD (length of diffusion) SA parameter in microns.

-sa5 *value*

LOD (length of diffusion) SA parameter in microns.

-sa6 *value*

LOD (length of diffusion) SA parameter in microns.

-sa7 *value*

LOD (length of diffusion) SA parameter in microns.

-sa8 *value*

LOD (length of diffusion) SA parameter in microns.

-sa9 *value*

LOD (length of diffusion) SA parameter in microns.

-sa10 *value*

LOD (length of diffusion) SA parameter in microns.

-sb1 *value*

LOD (length of diffusion) SB parameter in microns.

-sb2 *value*

LOD (length of diffusion) SB parameter in microns.

-sb3 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb4 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb5 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb6 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb7 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb8 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb9 ***value***

LOD (length of diffusion) SB parameter in microns.

-sb10 ***value***

LOD (length of diffusion) SB parameter in microns.

-sw1 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw2 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw3 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw4 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw5 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw6 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw7 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw8 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw9 ***value***

LOD (length of diffusion) SW parameter in microns.

-sw10 ***value***

LOD (length of diffusion) SW parameter in microns.

-sca ***value***

WPE (well proximity effect) SCA parameter.

-scb ***value***

WPE (well proximity effect) SCB parameter.

-scc ***value***

WPE (well proximity effect) SCC parameter.

-sc ***value***

WPE (well proximity effect) SC parameter.

-mulvsat ***value***

The mulvsat value is supplied to facilitate efficient modeling of mismatch, local variation, and mechanical stress and proximity effects.

-factuo ***value***

The zero-field mobility prefactor for certain device models.

-mulu0 ***value***

Multiplier for mu (electron mobility). The default is 1.0.

-mulid0 ***value***

The scaling factor for drain current. The default is 1.0.

-nrd ***value***

Resistance of drain terminal in squares.

-nrs ***value***

Resistance of source terminal in squares.

-pd ***value***

Perimeter of the source diffusion in microns.

-ps ***value***

Perimeter of the drain diffusion in microns.

-bvs_gain ***value***

This parameter is for a device model with a behavioral voltage source to model drain-induced barrier

lowering (DIBL) variations. It represents the gain of the behavioral voltage source associated with the transistor.

-nfin *value*

Finfet number of fins per finger.

-tfin *value*

Finfet transistor fin thickness in microns

-d *value*

Finfet transistor diameter of cylinder.

-fpitch *value*

Finfet transistor fin pitch in microns.

-aseo *value*

Finfet transistor source to substrate overlap area in square microns.

-adeo *value*

Finfet transistor drain to substrate overlap area in square microns.

-pseo *value*

Finfet transistor perimeter of source to substrate overlap region in microns.

-pdeo *value*

Finfet transistor perimeter of drain to substrate overlap region in microns.

-asej *value*

Finfet transistor source junction area in square microns.

-adej *value*

Finfet transistor drain junction area in square microns.

-psej *value*

Finfet transistor source junction perimeter in microns.

-pdej *value*

Finfet transistor drain junction perimeter in microns.

-lrsd *value*

Finfet transistor length of source/drain in microns.

-xl *value*

Finfet transistor L offset of channel length.

-delvtrand *value*

Finfet transistor parameter.

-u0mult *value*

Finfet transistor parameter.

-ids0mult *value*

Finfet transistor parameter.

-cgsp *value*

Finfet transistor constant gate to source fringe capacitance.

-cgdp *value*

Finfet transistor constant gate to drain fringe capacitance.

-cdsp *value*

Finfet transistor constant drain to source fringe capacitance.

-ngcon *value*

Finfet transistor number of gate contacts.

-name *parameter_name*

Used only with the **-value** parameter to specify the value of a specific parameter. One command can specify only one name value pair.

-value *value*

Used only with the **-name** option to specify the value of a specific parameter. One command can specify only one name value pair.

-quiet

Suppresses warning messages.

object_list

Specifies a list of transistors or parasitic devices affected by the command.

DESCRIPTION

This command sets one or more parameters for specified transistors in the design, overriding the parameters in the technology definition. The object list contains a list of transistors or parasitic devices affected by the command.

Changing any parameter changes the design and requires you to run the **check_design** and **trace_paths** commands to get analysis results, even if those commands have been run previously.

To view the parameter settings for a transistor, use the **report_attribute** or **get_attribute** command.

For example,

```
nt_shell> report_attribute -class cell \
          -application [get_cells Xreg51.X3.Mp0]
```

COMMAND PHASING

The **set_transistor_parameter** command can only be executed between the **link_design** command and the **check_design** command. To use the **set_transistor_parameter** command after the **check_design** command, you must first execute the **reset_design** command without any options.

EXAMPLES

The following command sets the length and width parameters for transistor Xreg51.X3.Mp0.

```
nt_shell> set_transistor_parameter -width 4.0 \
          -length 6.6 [get_cells Xreg51.X3.Mp0]
```

In the following example, the two instance parameters instpa and insptb are set to the values 0.2 and 0.3. This setting applies to all the cells found by the command get_cells Xreg51.x3

```
nt-shell> set_transistor_parameter -name instpa -value 0.2 \
          [get_cells Xreg51.x3]
nt-shell> set_transistor_parameter -name instpb -value 0.3 \
          [get_cells Xreg51.x3]
```

SEE ALSO

```
get_attribute(2)
report_attribute(2)
set_transistor_direction(2)
set_transistor_drive_factor(2)
mos_enable_lod(3)
mos_enable_wpe(3)
```

set_transition_coefficients

Modifies the output transition times of delay arcs with the specified trigger transistor gate pins.

SYNTAX

```
status set_transition_coefficients
   [-min]
   [-max]
   [-min_clock]
   [-max_clock]
   [-transition transition_factor]
   [-offset offset_value]
   objects
```

Data Types

transition_factor	float
offset_value	float
objects	objects

ARGUMENTS

-min

Modifies transition times for minimum delay (short data path) analysis.

-max

Modifies transition times for maximum delay (long data path) analysis.

-min_clock

Modifies transition times for minimum-clock (short clock path) analysis.

-max_clock

Modifies transition times for maximum-clock (long clock path) analysis.

-transition *transition_factor*

The transition factor, a positive number or zero. NanoTime multiplies the original calculated output

transition time by this factor to obtain the new transition time. The default is 1.0.

-offset offset_value

The offset value, a positive or negative number. NanoTime adds this number to the existing calculated output transition time (adjusted by the transition factor, if any) to obtain the new transition time. The default offset value is zero.

objects

The list of transistor gate pins which select a set of delay arcs which are affected by this command. This list defines a set of delay calculation trigger pins. The output transitions of delay arcs driven by these trigger pins use the modified transition times.

DESCRIPTION

The **set_transition_coefficients** command modifies the transition times at specified locations in the design. At each specified trigger transistor gate pin, NanoTime multiplies the calculated output transition time by a specified factor, then adds a specified offset value to obtain a new transition time for that location. This feature can be used to adjust transition times for effects not already modeled by the operating conditions.

In the command, you must specify the types of timing arcs affected by the command (min data, max data, min clock, or max clock) and a list of trigger transistor gate pins to be modified. You must also specify the multiplication factor or offset value, or both.

If the result of modification is a negative transition time, NanoTime uses a transition time of zero.

To find out about transition coefficients that have been set, use the **report_attribute** or **get_attribute** command.

To remove transition coefficients that have been set, use the **remove_transition_coefficients** command.

COMMAND PHASING

The **set_transition_coefficients** command is typically executed between the "netlist-linked" and "topology-checked" states (in other words, between the **link_design** and **check_design** commands). If you use the **set_transition_coefficients** command after the **check_design** command, the NanoTime tool transitions back to the "topology-checked" state and discards all commands and operations performed after the **check_design** command.

EXAMPLES

The following command modifies the output transition time for max-path delay checking for the delay arc whose trigger is transistor gate pin Xareg.D[53]. NanoTime calculates the output transition time, multiplies

the result by 1.2 and adds an offset of 0.1. If the original calculated output transition time is 2.0 time units, the final result used by NanoTime for analysis is 2.5 time units.

```
nt_shell> set_transition_coefficients -max \
          -transition 1.2 -offset 0.1 \
          [get_pins Xareg.D[53]]
```

SEE ALSO

get_attribute(2)
remove_delay_coefficients(2)
remove_transition_coefficients(2)
report_attribute(2)
set_delay_coefficients(2)

set_variation_parameters

Sets the parameters needed to compute variation of an arc delay used during path tracing and reporting.

SYNTAX

```
status set_variation_parameters
   -type  type
   [-input_transition values]
   -variation valuesfP
   [-transition_variation values]
   [-min | -max ]
   [-width value]
   [-length value]
   [-nfin num]
   [-voltage volt]
   [-transistor_model model]
   [-wrapper_parameters list]
   [-series2 factor2]
   [-series3 factor3]
   [-series4 factor4]
```

Data Types

```
   type             string
   values           list of floats
   num              integer
   volt             float
   model            string
   list             list
   factor2          float
   factor3          float
   factor4          float
```

ARGUMENTS

-type *type*

The type of variation being defined. Valid values are: **nmos**, **pmos**, **dcs**, **dds**, **ntm_dds**, **libcell**, and **default**.

-variation *values*

If the variable timing_enable_slew_variation is false then this is a single one sigma variation to apply for this type of delay. If the variable set_timing_enable_slew_variation is true, then this is a list of one sigma variation values that correspond to the values specified by -input_transition option.

-input_transistion *values*

This can only be used if the timing_enable_slew_variation variable is set to true. It then specifies the indices to be used at an input transition when computing delay and output transition variation.

-transition_variation *values*

This can only be used if the timing_enable_slew_variation variable is set to true. It then specifies the amount of variation at the output slope as a function of input slope specified by the -input_transition option. In this mode the size of the lists in input_transition, variation, and transition_variation must be equal.

-min

Specifies that the defined variation only applies during minimum delay path tracing and delay calculation.

-max

Specifies that the defined variation only applies during maximum delay path tracing and delay calculation.

-width *value*

The reference width, in microns, for the transistor whose variation is being described. Variation for other widths can be derived from a reference transistor.

Valid only with types **pmos** and **nmos**. Only one of the **-width** or **-nfin** options can be specified.

-length *value*

The reference length, in microns, for the transistor whose variation is being described. It is necessary to enumerate each unique transistor length with a separate command. If no length is specified, this command applies to all transistor lengths.

Valid only with types **pmos** and **nmos**.

-nfin *num*

The reference number of fins for the transistor whose variation is being described. Variation for other numbers of fins can be derived from a reference transistor.

Valid only with types **pmos** and **nmos**. Only one of the **-width** or **-nfin** options can be specified.

-voltage *volt*

The reference voltage for the transistor whose variation is being described. It is necessary to enumerate each unique voltage with a separate command. If no voltage is specified, this command applies to all voltages.

Valid only with types **pmos** and **nmos**.

-transistor_model *model*

The reference model for the transistor whose variation is being described. It is necessary to enumerate

each unique model with a separate command. If no model is specified, this command applies to all models for that type of transistor (**nmos** or **pmos**).

Valid only with types **pmos** and **nmos**.

-wrapper_parameters ***list***

For macro model transistors, the list of wrapper subcircuit parameter names and the value which must be matched. The name and value should be paired with =. For example, {param1=1.0 param2=0.5}.

Valid only with types **pmos** and **nmos**.

-series2 ***factor2***

The factor used to account for a chain of two series devices. If a chain of devices is used, the variation is reduced because of the interaction of the devices, although the other devices in the chain do not have the same impact as the switching device.

This value is the number of equivalent devices. The default is 1.0 and the allowable range is 1.0 to 2.0. Valid only with types **pmos** and **nmos**.

-series3 ***factor3***

The factor used to account for a chain of three series devices. This value is the number of equivalent devices. The default is 1.0 and the allowable range is 1.0 to 3.0.

Valid only with types **pmos** and **nmos**.

-series4 ***factor4***

Set the factor used to account for a chain of four or more series devices. This value is the number of equivalent devices. The default is 1.0 and the allowable range is 1.0 to 4.0.

Valid only with types **pmos** and **nmos**.

DESCRIPTION

The **set_variation_parameters** command specifies how variations for individual arc delays are computed. The nominal delay is computed for each delay in the path, then a variation equal to (nominal delay)*(value) is determined. The arc variation values are combined in a statistical fashion to compute the overall variation adjustment for a full path.

For more information and examples, see the *NanoTime User Guide.*

Specifying Variation for Transistors

For transistor-based delays, the variation is computed by using values from a reference transistor variation value. If a reference transistor is specified with its width or number of fins and variation values, other variation values are computed from this specification. This applies to all transistors with common lengths, voltages, and models. If a reference transistor is described without a length, voltage, or model, it is assumed to apply to all transistors for that parameter. For more precise calculation of variation, each combination of length, model, and voltage that generates unique behavior in regards to

variation should be modeled separately.

Transistor-based variations are specified with the **-type** option and arguments **nmos** or **pmos**. Either the **-width** or **-nfin** option is required. The **-length**, **-transistor_model**, **-voltage**, **-series2**, **-series3**, and **-series4** options are valid but optional.

The **-length**, **-width**, **-nfin**, **-transistor_model**, **-voltage**, **-series2**, **-series3**, and **-series4** options are valid only for types **nmos** and **pmos**. If you use these options with any other device type, NanoTime issues a CMDS-100 error and ignores the **set_variation_parameters** command.

Specifying Variation for Other Devices

For delays through dynamic delay simulation (DDS) regions, dynamic clock simulation (DCS) regions, or library cells, use the **-type** option with the argument **dds**, **ntm_dds**, **dcs**, or **libcell**. For these types of delay arcs, a single variation value is specified as the default for all arcs of the specified type. Type **dds** applies only to non-memory column DDS regions. Type **ntm_dds** applies to memory column DDS regions.

When the type dcs is used for dynamic clock simulation regions, the variation specified applies to all arcs within the region. If the **dcs_enable_detailed_path_reporting** variable is set to **true**, this variation value applies individually to each detailed arc through the DCS region. If the **dcs_enable_detailed_path_reporting** variable is **false**, the variation value is used for single arc delays through the region.

For library cell arcs, the **set_libcell_variation_parameters** command is available to assign variation values to specific library cells. Also, the variation information can come from LVF (Library Variation Format) table from a library file. The default variation set by the **set_variation_parameters -type libcell** command is considered only when the cell arcs have no variation set by the **set_libcell_variation_parameters** command or LVF table.

Specifying Default Variation

The **-type** option with the **default** argument is used to assign a variation value to any arc delay that is not described by one of the other options. This applies to any transistor-based delay which has a transistor which has not been modeled, or any other type of delay arc which does not have a variation value specified.

If NanoTime uses the default variation for the trigger device in a stage, the default variation is used to represent the entire stage. In this case, parameters such as the channel width or number of fins ar not taken into account for any of the transistors in the stage controlled by that trigger device. The best practice is to define transistor characteristics intentionally rather than using the default type.

COMMAND PHASING

The **set_variation_parameters** command must be used after the **check_design** command but before any path tracing commands (such as the **trace_paths** or **extract_model** commands).

SEE ALSO

`report_variation(2)`

```
set_libcell_variation_parameters(2)
timing_pocv_sigma(3)
tech_pocv_match_length_pct(3)
tech_pocv_match_voltage_pct(3)
dcs_enable_detailed_path_reporting(3)
timing_enable_slew_variation(3)
```

set_voltage

Sets the voltage of a power supply net.

SYNTAX

```
status set_voltage
   [-min min_corner_voltage]
   [-max max_corner_voltage]
   [-min_clock min_corner_clock_voltage]
   [-max_clock max_corner_clock_voltage]
   voltage
   net_list
```

Data Types

min_corner_voltage	float
max_corner_voltage	float
min_corner_clock_voltage	float
max_corner_clock_voltage	float
voltage	float
net_list	list

ARGUMENTS

-min *min_corner_voltage*

Specifies the voltage used to calculate minimum (shortest-path) delays. If not specified, the nominal value is used. This option requires a NanoTime Ultra license and must be specified with all four **-max/-min** options.

-max *max_corner_voltage*

Specifies the voltage used to calculate maximum (longest-path) delays. If not specified, the nominal value is used. This option requires a NanoTime Ultra license and must be specified with all four **-max/-min** options.

-min_clock *min_corner_clock_voltage*

Specifies the voltage used to calculate minimum delays for clock paths. If not specified, the nominal value is used. This option requires a NanoTime Ultra license and must be specified with all four **-max/-min** options.

-max_clock ***max_corner_clock_voltage***

Specifies the voltage used to calculate maximum delays for clock paths. If not specified, the nominal value is used. This option requires a NanoTime Ultra license and must be specified with all four **-max/-min** options.

voltage

Specifies the nominal voltage on the power supply net.

net_list

Specifies a list of power supply nets.

DESCRIPTION

The **set_voltage** command sets the voltage of specified power supply nets. Setting a voltage on a power supply net overrides any previous voltage setting for that net.

Before setting the voltage of a net, the net must be defined to be a power supply net, either implicitly at link time (as determined by the **link_vdd_alias** variable) or by using the **set_supply_net** command.

To determine the voltage of a net, use the **get_attribute** command.

The **-min**, **-max**, **-min_clock**, and **-max_clock** options cause the specified voltages to be used for the min data, max data, min clock, and max clock corners. Use these when performing multicorner analysis. All four corners must be specified. The input voltage swing in this situation is controlled globally by the value of the **oc_global_voltage** variable, or can be set on specific output ports using the **set_input_transition -rail_voltage** command. A model created under these conditions uses the delay values produced by the **-min** and **-max** options.

For timing model creation, NanoTime uses the highest design voltage for setting the nominal voltage in the model. However, you can change this behavior by setting the **model_apply_oc_global_voltage** variable to **true**, in which case the model nominal voltage is set to the value in the **oc_global_voltage** variable.

Virtual Supply Voltage

When the design contains virtual supply net and power switch devices, the voltage on the virtual supply nets can be different from the true supply voltage due to IR drop across the power switch transistors. To account for this effect, use the **set_voltage** command to specify the actual voltage on a virtual supply net if it is different from the true supply voltage.

If you do not set a voltage value on a virtual supply net, NanoTime includes the power switch devices in each relevant stage simulation in an attempt to account for the IR drop. However, this is only an approximation because the full IR drop depends on the dynamic circuit behavior of all stages connected to the power switch transistors.

COMMAND PHASING

The **set_voltage** command is typically executed in the "netlist-linked" state (between the **link_design** command and the **check_topology** command). If you execute it later, NanoTime transitions back to

the "netlist-linked" state and discards all operations performed after the **check_topology** command.

EXAMPLES

The following command sets the voltage of net vdd1 to 1.2 volts:

```
nt_shell> set_voltage 1.2 vdd1
```

The following command reports the voltage attribute of the vdd1 net:

```
nt_shell> get_attribute -class net vdd1 voltage
1.200000
```

The following command sets the voltages of the vdd1 net for multicorner analysis:

```
nt_shell> set_voltage -min 1.3 -max 1.1 -min_clock 1.25 -max_clock 1.15 1.2 vdd1
```

SEE ALSO

get_attribute(2)
set_technology(2)
set_supply_net(2)
link_vdd_alias(3)
oc_global_voltage(3)
model_apply_oc_global_voltage(3)

setenv

Sets the value of a system environment variable.

SYNTAX

```
string setenv variable_name new_value

string variable_name


string new_value
```

ARGUMENTS

variable_name

Names of the system environment variable to set.

new_value

Specifies the new value for the system environment variable.

DESCRIPTION

The **setenv** command sets the specified system environment *variable_name* to the *new_value* within the application. If the variable is not defined in the environment, the environment variable is created. The **setenv** command returns the new value of *variable_name*. To develop scripts that interact with the invoking shell, use **getenv** and **setenv**.

Environment variables are stored in the Tcl array variable **env**. The environment commands **getenv**, **setenv**, and **printenv** are convenience functions to interact with this array.

The **setenv** command sets the value of a variable only within the process of your current application. Child processes initiated from the application using the **exec** command after a usage of **setenv** inherit the new variable value. However, these new values are not exported to the parent process. Further, if you set

an environment variable using the appropriate system command in a shell you invoke using the **exec** command, that value is not reflected in the current application.

EXAMPLES

The following example changes the default printer.

```
shell> getenv PRINTER
laser1
shell> setenv PRINTER "laser3"
laser3
shell> getenv PRINTER
laser3
```

SEE ALSO

exec(2)
getenv(2)
unsettenv(2)
printenv(2)
printvar(2)
set(2)
sh(2)
unset(2)

sh

Executes a command in a child process.

SYNTAX

```
string sh [args]

string args
```

ARGUMENTS

args

Command and arguments that you want to execute in the child process.

DESCRIPTION

This is very similar to the **exec** command. However, file name expansion is performed on the arguments. Remember that quoting and grouping is in terms of Tcl. Arguments which contain spaces will need to be grouped with double quotes or curly braces. Tcl special characters which are being passed to system commands will need to be quoted and/or escaped for Tcl. See the examples below.

EXAMPLES

This example shows how you can remove files with a wildcard.

```
prompt> ls aaa*
aaa1      aaa2      aaa3
prompt> sh rm aaa*
prompt> ls aaa*
```

```
Error: aaa*: No such file or directory
       Use error_info for more info. (CMD-013)
```

This example shows how to grep some files for a regular expression which contains spaces and Tcl special characters:

```
prompt> exec cat test3.out
blah blah blah
blah blah blah c blah
input [1:0] A;
output [2:0] B;
prompt>
prompt> sh egrep -v {[ ]+c[ ]+} test3.out
blah blah blah
prompt>
prompt> sh egrep {t[ ]+\[} test3.out
input [1:0] A;
output [2:0] B;
```

SEE ALSO

exec(2)

sh_list_key_bindings

Displays all the key bindings and the edit mode of the current shell session. This variable is for use in Tcl mode only.

SYNTAX

```
status sh_list_key_bindings
   [-nosplit]
```

ARGUMENTS

-nosplit

Indicates that lines are not to be split when column fields overflow.

DESCRIPTION

The **sh_list_key_bindings** command displays current key bindings and the edit mode. To change the edit mode, use the **set_cle_options** command.

The text CTRL+K is read as `Control+K' and describes the character produced when the Ctrl key and the k key are pressed at the same time.

The text META+K is read as `Meta+K' and describes the character produced when the Meta key and k key are pressed at the same time. The Meta key is labeled ALT on many keyboards. On keyboards with two keys labeled ALT (usually to either side of the space bar), the ALT on the left side is generally set to work as a Meta key. The ALT key on the right might also be configured to work as a Meta key, but it might be configured as some other modifier, such as a Compose key for typing accented characters.

If you do not have a Meta or ALT key, or another key working as a Meta key, the identical keystroke can be generated by pressing the Esc key then the k key. Either process is known as metafying the k key.

In vi mode, pressing the Esc key changes the edit mode to alternate (command) mode. Alternate key bindings work only in vi alternate (command) mode.

In vi mode, if arrow keys are used while in input mode then the mode is automatically changed to alternate (command). Arrow keys should not be used to move around in the line while in input mode. If any mistake is made while entering text in input mode, first press the Esc key to return to alternate mode, then relocate the cursor to the position of the error.

COMMAND PHASING

The **sh_list_key_bindings** command is not phase-restricted.

EXAMPLES

SEE ALSO

```
sh_enable_line_editing(3)
set_cle_options(2)
```

sizeof_collection

Returns the number of objects in a collection.

SYNTAX

```
int sizeof_collection
   coll_A
```

Data Types

```
   coll_A      collection
```

ARGUMENTS

coll_A

Specifies the collection of interest. If the empty collection (empty string) is used for the *coll_A* argument, the command returns 0.

DESCRIPTION

The **sizeof_collection** command is an efficient mechanism for determining the number of objects in a collection.

COMMAND PHASING

The **sizeof_collection** command can be executed at any time.

EXAMPLES

The following example determines how many objects match a pattern and filter in a **get_cells** command.

```
nt_shell> echo "Number of hierarchical cells: \
                  [sizeof_collection [get_cells * -hier \
                     -filter "is_hierarchical == true"]]"
Number of hierarchical cells is: 10
```

The following example shows what happens when the argument for the **sizeof_collection** command results in an empty collection.

```
nt_shell> set s1 [get_cells *]
{"u1", "u2", "u3"}
nt_shell> set ssize [filter_collection $s1 "area < 0"]
nt_shell> echo "Cells with area < 0: [sizeof_collection $ssize]"
Cells with area < 0: 0
```

SEE ALSO

collections(2)
filter_collection(2)

sort_collection

Sorts a collection based on one or more attributes, resulting in a new, sorted collection.

SYNTAX

```
collection sort_collection
   [-descending]
   [-dictionary]
   coll_A
   criteria
```

Data Types

coll_A	collection
criteria	list

ARGUMENTS

-descending

Sorts the collection in descending order. By default, the sort is in ascending order.

-dictionary

Sorts the collection in dictionary order.

coll_A

Specifies the collection to be sorted.

criteria

Specifies a list of one or more attributes to use as sort keys.

DESCRIPTION

Use the **sort_collection** command to order the objects in a collection based on one or more attributes. For example, to get a collection of leaf cells increasing alphabetically, followed by hierarchical cells increasing alphabetically, sort the collection of cells using the **is_hierarchical** and **full_name** attributes as criteria.

In an ascending sort on Boolean attributes, objects whose attributes have a value of **false** precede objects with attributes set to **true**. In the case of a sparse attribute, objects that have the attribute come first, followed by objects that do not have the attribute.

Sorts are ascending by default. The **-descending** option reverses the order of the objects.

The **-dictionary** option primarily affects numeric characters. In a standard ascending sort, objects with numbers embedded in their names do not follow numerical order. The **-dictionary** option enforces numerical order. For example, if there is a set of objects named port0, port1, port2, and so on, a standard ascending sort orders them as follows:

```
port0
port1
port11
port12
port13
...
port18
port19
port2
port20
port 21
...
```

If you use the **-dictionary** option, the sort order is as follows:

```
port0
port1
port2
port3
port4
...
port9
port10
port11
port12
...
```

The **-dictionary** option and **-descending** options can be used together.

COMMAND PHASING

The **sort_collection** command can be executed at any time.

EXAMPLES

The following example sorts a collection of cells based on hierarchy and adds a second key to list them alphabetically. In this example, cells i1 and i2 are hierarchical, and o1 and o2 are leaf cells. Because **is_hierarchical** is a Boolean attribute, those objects with the attribute set to **false** are listed first in the

sorted collection.

```
nt_shell> set zcells [get_cells {o2 i2 o1 i1}]
{"o1", "i2", "o1", "i1"}
nt_shell> set zsort [sort_collection $zc {is_hierarchical full_name}]
{"o1", "o2", "i1", "i2"}
```

SEE ALSO

collections(2)

source

Read a file and evaluate it as a Tcl script.

SYNTAX

```
string source [-echo] [-verbose] [-continue_on_error] file

string file
```

ARGUMENTS

-echo

Echoes each command as it is executed. Note that this option is a non-standard extension to Tcl.

-verbose

Displays the result of each command executed. Note that error messages are displayed regardless. Also note that this option is a non-standard extension to Tcl.

-continue_on_error

Don't stop script on errors. Similar to setting the shell variable sh_continue_on_error to true, but only applies to this particular script.

file

Script file to read.

DESCRIPTION

The **source** command takes the contents of the specified *file* and passes it to the command interpreter as a text script. The result of the source command is the result of the last command executed from the file. If an error occurs in evaluating the contents of the script, then the **source** command returns that error. If a return command is invoked from within the file, the remainder of the file is skipped and the source command returns normally with the result from the return command.

By default, source works quietly, like UNIX. It is possible to get various other intermediate information from the source command using the **-echo** and **-verbose** options. The **-echo** option echoes each command as it appears in the script. The **-verbose** option echoes the result of each command after execution.

NOTE: To emulate the behavior of the dc_shell **include** command, use both of these options.

The file name can be a fully expanded file name and can begin with a tilde. Under normal circumstances, the file is searched for based only on what you typed. However, if the system variable sh_source_uses_search_path is set to "true", the file is searched for based on the path established with the search_path variable.

The **source** command supports several file formats. The *file* can be a simple ascii script file, an ascii script file compressed with gzip, or a Tcl bytecode script, created by TclPro Compiler. The **-echo** and **-verbose** options are ignored for gzip formatted files.

EXAMPLES

This example reads in a script of aliases:

```
prompt> source -echo aliases.tcl
alias q quit
alias hv {help -verbose}
alias include {source -echo -verbose}
prompt>
```

SEE ALSO

search_path(3)
sh_source_uses_search_path(3)
sh_continue_on_error(3)

suppress_message

Disables printing of one or more informational or warning messages.

SYNTAX

```
string suppress_message [message_list]

list message_list
```

ARGUMENTS

message_list

A list of messages to suppress.

DESCRIPTION

The **suppress_message** command provides a mechanism to disable the printing of messages. You can suppress only informational and warning messages. The result of **suppress_message** is always the empty string.

A given message can be suppressed more than once. So, a message must be unsuppressed (using **unsuppress_message**) as many times as it was suppressed in order for it to be enabled. The **print_suppressed_messages** command displays the currently suppressed messages.

EXAMPLES

When the argument to the **unalias** command does not match any existing aliases, the CMD-029 warning message displays. This example shows how to suppress the CMD-029 message:

```
prompt> unalias q*
Warning: no aliases matched 'q*' (CMD-029)
prompt> suppress_message CMD-029
prompt> unalias q*
prompt>
```

SEE ALSO

print_suppressed_messages(2)
unsuppress_message(2)
get_message_ids(2)
set_message_info(2)

suspend_licenses

Checks in NanoTime licenses, waits, then checks back out NanoTime licenses.

SYNTAX

```
status suspend_licenses
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

The **suspend_licenses** command is used to suspend a NanoTime session, returning the NanoTime licenses to the license pool. After releasing the licenses, the **suspend_licenses** command waits to receive a carriage return, at which time it checks out the licenses which it had released, and the NanoTime session continues as normal. The NanoTime process remains alive on the host machine throughout the command.

Here is an example usage of the **suspend_licenses** command:

```
nt_shell> suspend_licenses
Checked in license 'NanoTime'
Hit RETURN to resume...
Checked out license 'NanoTime'
1
```

COMMAND PHASING

The **suspend_licenses** command is not phase-restricted.

trace_paths

Traces timing paths in the current design and saves the worst-case paths into a database for reporting and other processing.

SYNTAX

```
status trace_paths
   [-max_only]
   [-min_only]
   [-clock_only]
   [-full_path_enumeration]
   [-worst_fanin_path]
   [-npaths number_of_paths]
   [-keep_paths_within time_value]
   [-pbsa]
   [-pocv]
```

Data Types

```
   number_of_paths    integer
   time_value         float
```

ARGUMENTS

-max_only

Traces maximum delay (longest) paths and skips the tracing of minimum delay (shortest) paths.

-min_only

Traces minimum delay (shortest) paths and skips the tracing of maximum delay (longest) paths.

-clock_only

Traces clock network paths and skips the tracing of maximum delay and minimum delay data paths. Using this option saves time when you only want to check the clock network timing.

-full_path_enumeration

Traces all paths using full path enumeration, without pruning. To save the additional information in the path database, set the **-npaths** option to a value larger than 1. Due to the large runtime requirements, this option is practical only when used with the **set_find_path** command to limit the scope of the

design being considered in the analysis.

-worst_fanin_path

Keeps only worst timing paths in the fanin to each endpoint, irrespective of startpoints. In the absence of this option, NanoTime keeps the worst path from each startpoint to each endpoint. Using this option keeps fewer paths and reduces runtime.

-npaths ***number_of_paths***

Specifies the maximum number of paths to save in the path database for each startpoint-endpoint pair. The default setting is 1. Specify a larger number to save more paths, but use a value that is small enough to keep the runtime and memory usage reasonable. This option can only be used together with the **-keep_paths_within** option.

-keep_paths_within ***time_value***

Keeps timing paths in the path database that have a path delay within the *time_value* of the worst path delay of each startpoint-endpoint pair. By default, **-keep_paths_within** is set to 0.0 and NanoTime keeps only the single worst path per startpoint-endpoint pair. To keep more paths, set **-keep_paths_within** to a time value larger than 0.0 and set the **-npaths** option to a number larger than 1. This puts more timing information in the path database, but also increases the runtime and memory usage.

-pbsa

Invokes path-based slack adjustment, which adjusts the calculated path delays based on the lengths of the path segments traced in the timing check. A NanoTime Ultra license is required to use this feature.

-pocv

Invokes random on-chip variation path delay and slack adjustment, which adjusts the calculated path delays based on the timing variation from each path arc. A NanoTime Ultra license is required to use this feature. For more information, see the *NanoTime User Guide*.

DESCRIPTION

The **trace_paths** command traces through the design, path by path, and saves detailed information on the worst-case timing paths into a database. When NanoTime completes execution of the command, you can use the **report_paths** command to generate various types of reports on the design timing. The **report_paths** command gets the timing information from the database without performing any additional path tracing or simulation. You can also use the **get_timing_paths** command to report on a wide range of information in the database.

Before you can use the **trace_paths** command, you must prepare and check the design data by using the commands **link_design**, **check_topology**, and **check_design**. You can optionally restrict the scope of the analysis with the **set_find_path** command, which specifies the startpoints, throughpoints, endpoints, clocks, and edge types to be considered by the **trace_paths** command. If used, the **set_find_path** command must be run before the **check_design** command.

If you need to repeat the path tracing process with different settings, you can use the **reset_design -paths** command to reset the database, and then repeat the analysis.

The **trace_paths** command breaks up the design into discrete units, simulates those units to characterize their timing, and traces paths through the design while analyzing the path timing behavior. It determines which paths have the most critical timing and saves the results into the path database. The command reports its progress as it performs different steps of the timing analysis. For example:

```
nt_shell> trace_paths -max_only
Building simulation units ...
Built 462 simulation units with a total of 15399
  instances
Done building simulation units
Performing max clock search
Updating clock arrivals with pulse properties
...
```

During the path tracing process, NanoTime uses several different techniques to reduce the number of paths that must be traced and the number saved in the database. These techniques reduce the total runtime used for path tracing, while keeping enough information for detailed timing reports.

The **trace_paths** command offers several options with respect to the types of paths analyzed and the numbers of paths saved in the database. In addition, options such as exhaustive path tracing are available for debugging purposes.

Restricting the Path Types

By default, the **trace_paths** command traces both minimum delay (shortest) data paths and maximum delay (longest) data paths. If you only want to analyze one type of path, use the **-min_only** or **-max_only** option to restrict path searching to that type of path. This reduces the runtime.

The **trace_paths** command traces clock networks, in addition to minimum delay and maximum delay data paths, to determine the amount of timing slack of each data path. It does this even if you use the **-min_only** or **-max_only** option. If you only want to analyze the timing of the clock network, without considering data paths at all, use the **-clock_only** option.

Pruning Paths and Saving Worst-Case Paths

The goal of path tracing is to find the longest paths, shortest paths, and worst-case paths (those with the least slack) and to save information on those paths into a database from which you can generate reports with the **report_paths** command. The **set_find_path** command and the **trace_paths** command options let you control the amount of information traced and saved, so you can get all the information you need, while keeping the runtime and memory usage reasonable.

As NanoTime traverses the tree of paths in the design, it prunes the branches that it determines to be not the worst-case paths. For example, when it traces a new path to a point in a previously calculated path, and the delay to that point is better than the delay calculated in the previous path, NanoTime immediately determines that the new path being traced is not the worst-case path. Therefore, it stops tracing that path and starts tracing the next path. Pruning reduces the runtime because it eliminates unnecessary tracing.

By default, NanoTime keeps the worst path from each startpoint to each endpoint. However, if you use the **-worst_fanin_path** option, it keeps only the worst timing paths in the fanin to each endpoint, irrespective of startpoints. This results in fewer paths traced and saved because there is only one path saved in the fanin to each endpoint, rather than one for every startpoint-endpoint pair.

The **-keep_paths_within** and **-npaths** options let you save more paths at the cost of more runtime and memory. By default, NanoTime keeps the single worst path per startpoint-endpoint pair. However,

there could be many more paths that are nearly as bad as the worst one. To keep these almost-as-bad timing paths in the database, set the **-keep_paths_within** option to a value slightly larger than the default of 0.0, and set the **-npaths** option to a value larger than 1. For example:

```
nt_shell> trace_paths -keep_paths_within 0.2 -npaths 3
```

In this example, for each startpoint-endpoint pair, NanoTime keeps all paths that have a slack within 0.2 time units of the worst path, up to a maximum of three paths per startpoint-endpoint pair. If a startpoint-endpoint pair has many worst-case paths with a slack within 0.2 time units of each other, only the worst three of them are saved in the database. Be sure to set the **-npaths** option to a number small enough to ensure a reasonable runtime.

You cannot choose to save path delays based on startpoints or throughpoints. Even if you increase the number of paths saved for each startpoint-endpoint pair, you cannot guarantee that a specific path of interest will be available for later reporting. If you want to ensure that a specific path is included in the path tracing database, you must use the **set_find_path** command with the appropriate **-from** or **-through** options before you use the **check_design** command. In this case, path tracing is performed only on the specified paths; delays in the rest of the design are not saved in the path database.

To reduce memory usage, you can set an overall limit on the number of paths saved for the whole design. The following variables set the limits for different types of timing paths. (The variables are shown with their defaults.)

```
timing_max_delay_paths_saved 1000000
timing_max_slack_paths_saved 1000000
timing_min_delay_paths_saved 1000000
timing_min_slack_paths_saved 1000000
```

The "delay" paths are longest maximum delay paths and shortest minimum delay paths, irrespective of clock arrival times. The "slack" paths are the maximum delay and minimum delay paths that have the least slack, taking into account both the data and clock arrival times. The worst-delay and worst-slack paths are often the same, depending on the amount of clock skew among different paths.

For example, to limit the total number of maximum-delay paths saved in the database to 500 paths for the whole design:

nt_shell> **set timing_max_delay_paths_saved 500**

Path Tracing Parameters

Various aspects of path tracing behavior can be controlled by related variables and commands. Use the **set** command to set the variables before you use the **trace_paths** command.

The following variables control the determination of launch and corresponding capture edges throughout the design. The arguments shown here are the variable defaults.

```
timing_intersection_transparency true
timing_same_phase_gated_clock false
timing_default_phasing same_cycle
```

When evaluating a timing check whose capture clock period is a non-integer multiple of the launch clock period, NanoTime rounds the multiple to the nearest integer to determine the most restrictive clock cycles to compute the slack. This might result in NanoTime reporting an incorrect slack value.

The following commands control the determination of launch and corresponding capture edges for specific portions of the design.

```
set_phasing
set_no_same_phase_path
set_same_phase_path
```

The following variables control the types of information saved in the path database. The arguments shown here are the variable defaults.

```
timing_save_clock_paths true
timing_save_pin_arrival_and_transition false
timing_save_wire_delay true
```

The following variables control the behavior of path tracing with respect to three-state nodes. The arguments shown here are the variable defaults.

```
timing_turnoff_delay_max_fall 0
timing_turnoff_delay_max_rise 0
timing_turnoff_delay_min_fall 0
timing_turnoff_delay_min_rise 0
timing_turnoff_max_propagation false
timing_turnoff_min_propagation false
timing_turnoff_transition_max_fall .05
timing_turnoff_transition_max_rise .05
timing_turnoff_transition_min_fall .05
timing_turnoff_transition_min_rise .05
```

The following variables control other aspects of path tracing. The arguments shown here are the variable defaults.

```
trace_latch_error_recovery true
trace_show_stack_period 0
trace_search_depth_limit 1000000
trace_through_inputs true
trace_through_outputs true
trace_transparency_depth_closing_edge false
trace_transparency_depth_limit 1000000
trace_transparent_loop_checking false
transistor_stack_bidirectional_limit 2
transistor_stack_height_limit 10
```

Investigating Path Tracing

The **-full_path_enumeration** option is provided for investigating or debugging the path tracing process. It causes the **trace_paths** command to exhaustively trace all possible paths, without pruning. Using this option is the same as setting the **-keep_paths_within** option to a large number. To save the additional information in the path database, set the **-npaths** option to a value larger than 1.

Due to the large runtime requirements, this option is practical only when used with the **set_find_path** command to limit the scope of the design being considered in the analysis.

Reporting Paths in the Database

Use the **report_paths** command to generate reports on the paths in the path database. You can report all paths or just minimum delay, maximum delay, or clock paths. You can also specify the level

of detail in the report.

To extract information from the database that is not accessible with the **report_paths** command, use the **get_timing_paths** command to create a collection of path objects. Then use the **foreach_in_collection** and **get_attribute** commands to generate the reports.

Path-Based Slack Adjustment (PBSA)

The **-pbsa** option invokes path-based slack adjustment, which adjusts the calculated path delays according to the lengths of the path segments traced in the timing check. A timing analysis using PBSA results in less pessimism than setting global clock uncertainty values with the **set_clock_uncertainty** command, but requires more effort and runtime.

With the **-pbsa** option, NanoTime scales the global uncertainty settings set with the **set_clock_uncertainty** command using the KUsetup and KUhold multiplier settings, and it adjusts the calculated delays all timing paths using the remaining PBSA multiplier settings. The **-pbsa** option is also useful to perform clock reconvergence pessimism removal even if no multipliers are used. In the absence of the **-pbsa** option, the path tracing operation uses the global uncertainty settings made with the **set_clock_uncertainty** command without modification and ignores the PBSA parameter settings.

A NanoTime Ultra license is required to use path-based slack adjustment. For more information about PBSA, see the NanoTime User Guide or the man page for the **report_pbsa_calculation** command.

Parametric On-Chip Variation(POCV)

The **-pocv** option invokes parametric on-chip variation analysis, which accounts for local fluctuations in device behavior using statistical modeling techniques. You must define the variation analysis conditions by using commands such as the **set_variation_parameters** command and variables such as the **timing_pocv_sigma** variables.

A NanoTime Ultra license is required to use POCV. For more information, see the NanoTime User Guide.

COMMAND PHASING

The **trace_paths** command can only be executed in the "design-checked" state. After successful execution, the state is changed to "paths-traced."

EXAMPLES

The following command traces paths throughout the whole design (if the **set_find_path** command was not used previously) and saves information on the worst-case minimum delay path and the worst-case maximum delay path for each startpoint-endpoint pair.

```
nt_shell> trace_paths
```

The following example traces maximum delay paths that start from port IN1 and end on either port OUT1 or port OUT2. Upon completion, the path database contains the worst-case maximum delay path for each startpoint-endpoint pair (from IN1 to OUT1 and from IN1 to OUT2).

```
nt_shell> set_find_path \
```

```
            -from [get_ports IN1] \
            -to [get_ports {OUT1 OUT2}]
 ...
nt_shell> trace_paths -max_only
```

The following command traces maximum delay paths and saves information on up to three worst-case maximum delay paths for each startpoint-endpoint pair. The second-worst and third-worst paths are saved only if their slack values are within 0.5 time units of the worst path.

```
nt_shell> trace_paths -max_only \
            -keep_paths_within 0.5 \
            -npaths 3
```

SEE ALSO

get_timing_paths(2)
report_paths(2)
report_pbsa_calculation(2)
set_variation_parameters(2)
timing_pocv_sigma(3)
set_find_path(2)
timing_default_phasing(3)
timing_intersection_transparency(3)
timing_same_phase_gated_clock(3)
timing_save_clock_paths(3)
timing_save_pin_arrival_and_transition(3)
timing_save_wire_delay(3)
timing_turnoff_delay_max_fall(3)
timing_turnoff_max_propagation(3)
timing_turnoff_transition_max_fall(3)
trace_latch_error_recovery(3)
trace_search_depth_limit(3)
trace_show_stack_period(3)
trace_through_inputs(3)
trace_through_outputs(3)
trace_transparency_depth_closing_edge(3)
trace_transparency_depth_limit(3)
trace_transparent_loop_checking(3)
transistor_stack_bidirectional_limit(3)
transistor_stack_height_limit(3)

unalias

Removes one or more aliases.

SYNTAX

```
string unalias
   patterns
```

ARGUMENTS

patterns

Specifies the patterns to be matched. This argument can contain more than one pattern. Each pattern can be the name of a specific alias to be removed or a pattern containing the wildcard characters ***** and **%**, which match one or more aliases to be removed.

DESCRIPTION

The **unalias** command removes aliases created by the **alias** command.

EXAMPLES

The following command removes all aliases.

```
prompt> unalias *
```

The following command removes all aliases beginning with f, and the alias rt100.

```
prompt> unalias f* rt100
```

SEE ALSO

alias(2)

unsetenv

Removes a system environment variable.

SYNTAX

```
string getenv
   variable_name
```

Data Types

```
   variable_name      string
```

ARGUMENTS

variable_name

Specifies the name of the environment variable to be unset.

DESCRIPTION

The **unsetenv** command searches the system environment for the specified *variable_name* and removes variable from the environment. If the variable is not defined in the environment, the command returns a Tcl error. The command is catchable.

Environment variables are stored in the **env** Tcl array variable. The **unsetenv**, commands is a convenience function to interact with this array. It is equivalent to 'unset ::env(*variable_name*)'

The application you are running inherited the initial values for environment variables from its parent process (that is, the shell from which you invoked the application). If you unset the variable using the **unsetenv** command, you remove the variable value in the application and in any new child processes you initiate from the application using the **exec** command. However, the variable is still set in the parent process.

See the **set** and **unset** commands for information about working with non-environment variables.

EXAMPLES

In the following example, **unsetenv** remove the DISPLAY varible from the environment:

```
prompt> getenv DISPLAY
host:0
prompt> unsetenv DISPLAY
prompt> getenv DISPLAY
Error: can't read "::env(DISPLAY)": no such variable
        Use error_info for more info. (CMD-013)
```

SEE ALSO

```
catch(2)
exec(2)
printenv(2)
set(2)
unset(2)
setenv(2)
getenv(2)
```

unsuppress_message

Enables printing of one or more suppressed informational or suppressed warning messages.

SYNTAX

```
string unsuppress_message [messages]

list messages
```

ARGUMENTS

messages

A list of messages to enable.

DESCRIPTION

The **unsuppress_message** command provides a mechanism to re-enable the printing of messages which have been suppressed using **suppress_message**. You can suppress only informational and warning messages, so the **unsuppress_message** command is only useful for informational and warning messages. The result of **unsuppress_message** is always the empty string.

You can suppress a given message more than once. So, you must unsuppress a message as many times as it was suppressed in order to enable it. The **print_suppressed_messages** command displays currently suppressed messages.

EXAMPLES

When the argument to the **unalias** command does not match any existing aliases, the CMD-029 warning message displays. This example shows how to re-enable the suppressed CMD-029 message. Assume that

there are no aliases beginning with 'q'.

```
prompt> unalias q*
prompt> unsuppress_message CMD-029
prompt> unalias q*
Warning: no aliases matched 'q*' (CMD-029)
```

SEE ALSO

print_suppressed_messages(2)
suppress_message(2)

update_noise

Generates noise values by using coupling capacitances and aggressor slopes.

SYNTAX

```
status update_noise
```

ARGUMENTS

This command has no arguments.

DESCRIPTION

This command is used to generate noise values. It must be run after path tracing has been completed so that NanoTime can determine the slope values on nets. Noise analysis takes into account any user-set noise margins and user exclusions of victim or aggressor nets.

COMMAND PHASING

The **update_noise** command can be executed any time after the **trace_paths**, **extract_model**, or **characterize_context** commands.

EXAMPLES

The following command generates noise values for the current design.

```
nt_shell> update_noise
```

SEE ALSO

report_noise(2)
report_noise_violation_sources(2)
set_si_noise_analysis(2)
remove_si_noise_analysis(2)
report_si_noise_analysis(2)
set_noise_margin(2)
si_enable_noise_analysis(3)
si_noise_margin_above_high(3)
si_noise_margin_above_low(3)
si_noise_margin_below_high(3)
si_noise_margin_below_low(3)
report_fanout_noise(2)
set_fanout_noise_threshold(2)
si_enable_noise_fanout_analysis(3)

which

Locates a file and displays its pathname.

SYNTAX

```
string which filename_list

list filename_list
```

ARGUMENTS

filename_list

List of files to locate.

DESCRIPTION

Displays the location of the specified files. This command uses the search_path to find the location of the files. This command can be a useful prelude to read_db or link_design, because it shows how these commands expand filenames. The **which** command can be used to verify that a file exists in the system.

If an absolute pathname is given, the command searches for the file in the given path and returns the full pathname of the file.

EXAMPLES

The following examples are based on the following search_path.

prompt> **set search_path "/u/foo /u/foo/test"**

The following command searches for the file name foo1 in the search_path.

```
prompt> which foo1
/u/foo/foo1
```

The following command searches for files foo2, foo3.

```
prompt> which {foo2 foo3}
/u/foo/test/foo2 /u/foo/test/foo3
```

The following command returns the full pathname.

```
prompt> which ~/test/designs/sub_design.db
/u/foo/test/designs/sub_design.db
```

SEE ALSO

link_design(2)
read_db(2)
search_path(3)

write_app_var

Writes a script to set the current variable values.

SYNTAX

```
string write_app_var
   -output file
   [-all | -only_changed_vars]
   [pattern]
```

Data Types

```
   file         string
   pattern      string
```

ARGUMENTS

-output *file*

Specifies the file to which to write the script.

-all

Writes the default values in addition to the current values of the variables.

-only_changed_vars

Writes only the changed variables. This is the default when no options are specified.

pattern

Writes the variables that match the specified *pattern*. The default is "*".

DESCRIPTION

The **write_app_var** command generates a Tcl script to set all application variables to their current

values. By default, variables set to their default values are not included in the script. You can force the default values to be included by specifying the **-all** option.

EXAMPLES

The following is an example of the **write_app_var** command:

```
prompt> write_app_var -output sh_settings.tcl sh*
```

SEE ALSO

get_app_var(2)
report_app_var(2)
set_app_var(2)

write_context

Writes the timing context of a list of instances into a script file.

SYNTAX

```
status write_context
   [-timing]
   [-environment]
   [-debug_paths]
   [-constant_inputs]
   [-output my_file]
   [-derive_file_name]
   [-prefix my_prefix]
   [-extension my_extension]
   [-separator my_separator]
   [-script_directory my_dir]
   cell_list
   in -0.75i
```

Data Types

```
   my_file              string
   my_prefix            string
   my_extension         string
   my_separator         string
   my_dir               string
   cell_list            list
```

ARGUMENTS

-timing

Writes only the timing information such as clocks, input delays, output delays, and timing exceptions.

-environment

Writes only the only environment-related information such as operating conditions (process, temperature, and voltage) and capacitive loads on input and output pins.

-debug_paths

Writes a file to the working directory that lists the specific worst-case timing paths in the design that

were used to generate the boundary constraints.

-constant_inputs

Writes only the logic constants on input pins of the characterized instance.

-output *my_file*

Specifies the name of the generated script file. You can use either the **-output** or **-derive_file_name** option, but not both. If you use neither, the script is written to standard output. You cannot use the **-prefix**, **-extension**, **- script_directory**, or **-separator** options with the **-output** option.

-derive_file_name

Derives the name of the generated script file from the instance name. If you use this option, you can also use the **-prefix**, **-extension**, **- script_directory**, and **-separator** options.

-prefix *my_prefix*

Specifies the prefix for the derived constraint file name. If this option is not used, no prefix is added to the derived file name. You cannot use the slash (/) or apostrophe (') characters in the prefix.

-extension *my_extension*

Specifies the extension for the derived constraint file name. If this option is not used, the default extension is ntsh.

-separator *my_separator*

Specifies a single character to use as a hierarchical separator in constructing the derived constraint file name generated for each instance. If this option is not used, the default is the at sign (@). You cannot use the slash (/) or backslash (\) characters as name separators.

-script_directory *my_dir*

Specifies the directory in which to write the constraint file. Valid only with the **-derive_file_name** option. The default is the current directory.

cell_list

The list of instances in the current design for which to write the context out as a script.

DESCRIPTION

The **write_context** command writes the timing context of the specified instances as a script in Synopsys Design Constraints format. The context must first be captured by using the **characterize_context** command.

When you execute the SDC commands on the subdesign, the analysis tool initializes its timing environment to the information characterized from the context.

COMMAND PHASING

The **write_context** command can only be executed after the **characterize_context** command.

EXAMPLES

The following commands derive the timing-related context information for instance I1 and write the context information to a Synopsys Design Constraints (SDC) file called I1.ntsh in the current directory.

```
nt_shell> characterize_context {I1}

nt_shell> write_context -derive_file_name {I1}
```

SEE ALSO

characterize_context(2)
extract_model(2)

write_device_parameters

Writes device parameter data annotated on the design into a new file.

SYNTAX

```
status write_device_parameters
   [-format file_format]
   [-divider character]
   file_name
```

Data Types

```
   file_format        string
   character          string
   file_name          string
```

ARGUMENTS

-format *file_format*

Specifies the format of the output device parameter file. The only allowed value is **DSPF** (Detailed Standard Parasitic Format). Specifying the format is optional.

-divider *character*

Specifies the character divider to use for hierarchical names.

file_name

The name of the device parameter data file written by the command.

DESCRIPTION

The **write_device_parameters** command writes all device parameter data annotated on the current design to the file specified in the *file_name* argument. The command supports the Detailed Standard Parasitic Format (DSPF).

The command writes only the device parameter data annotated on transistors, not parasitic data annotated on nets. To write the net parasitic data to a file, use the **write_parasitics** command.

COMMAND PHASING

The **write_device_parameters** command can be executed any time after the **link_design** command.

EXAMPLES

The following command writes the currently annotated device parameter data into a file called top.dspf.

```
nt_shell> write_device_parameters -format DSPF top.dspf
```

SEE ALSO

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

write_global_topology_submission

Writes a user lib_topology object into a file for submission to Synopsys for possible inclusion in the standard global topology library.

SYNTAX

```
status write_global_topology_submission
   [-directory dir_name]
   topo
```

Data Types

```
   dir_name          string
   topo              list
```

ARGUMENTS

-directory *dir_name*

Specifies the directory path for writing the global topology submission file.

topo

Specifies a collection containing the single *lib_topology* object to be submitted to Synopsys, collected with the **get_lib_topology** command.

DESCRIPTION

You can submit a custom *lib_topology* object to Synopsys for possible inclusion in the global topology library of a future release of NanoTime. In that case, the library topology will become available to all users of NanoTime.

If a *lib_topology* object is accepted into the global library, you will receive a confirmation.

Use the following procedure to submit *lib_topology* objects to the global library.

1. Prepare the topology pattern *file lib_topo_name*.sp according to these SPICE pattern guidelines:
   - All transistor sizes MUST be removed.
   - All transistor parameters MUST be removed.
   - All transistor names MUST have proprietary naming removed.
   - Transistor naming is encouraged according to function (for example, mnfbk1, mpffwd1).
   - Net naming is encouraged to indicate function (for example, master_latch_net, slave_latch_net, clock_net, clock_net_bar, etc.)
   - Use lowercase characters for all names.
   - Comments inside the SPICE pattern file are encouraged.
2. Prepare the *lib_topology* definition file *lib_topo_name*.libt according to these guidelines:
   - The **-description** option is required. Place an abstract summary of the object in this option string value. The **-version** option is not needed. The submission mode replaces the option string with the current NanoTime version string.
   - At least one **-action*** option is required. The **-action_pre_match_topology** action is expected.
   - All names in the action commands must match the pattern netlist. Lowercase names are recommended.
3. Prepare an optional schematic drawing according to these guidelines:
   - The schematic is for efficient recognition of topology structure and function. The only required elements in the drawing are the nets and transistors referenced by Tcl commands in the action packages.
   - All names used in the schematic must exactly match the names in the pattern netlist. You can create the schematic drawing using any method or process that generates a final PDF document. The final image must be publishable as an open source document free of any licensing or NDA requirements.
   - Name the document lib_topo_name.pdf and place it in the same directory with the other *lib_topology* files. It will be automatically included in the submission file created by the **write_global_topology_submission** command.
4. Create a test case for the library topology. The following must be included in the test case:
   - File lib_topo_name.v, a Verilog netlist to instantiate one instance of the lib_topology
   - File lib_topo_name.model, a direct-read transistor model data
   - File lib_topo_name.tech, a SPICE netlist containing "*nanosim tech" commands for the SPICE model
   - File lib_topo_name.tcl, a NanoTime Tcl script to link the design, match the lib_topology candidate, and analyze all the "through" paths and timing checks in the candidate topology. The test case Tcl script must time the candidate topology and then write out a lib_topology submission file.
5. Create the submission package.

   Execute the submission command as the last command of the test case Tcl script. The command

packages all the files with the "lib_topo_name" base name and any extension.

Package command example:

```
write_global_topology_submission \
   -directory  testcase_dir_path \
   "lib_topo_name"
```

NanoTime expects to find the following files in the testcase_dir_path:

Test case files

```
lib_topo_name_test.tcl
lib_topo_name_test.sp
lib_topo_name_test.rpt
```

Topology files

```
lib_topo_name.sp
lib_topo_name.libt
```

Ntlib file

```
lib_name.ntlib
```

Schematic PDF (optional)

```
lib_topo_name.pdf
```

NanoTime creates the following files in testcase_dir_path:

```
Generated submission file: lib_topo_name.submission.gz
Miscellaneous runtime files from the test case run
```

6. Submit the *lib_topology* tarfile to Synopsys.

COMMAND PHASING

The **write_global_topology_submission** command is not phase-restricted.

EXAMPLES

The following command writes a representation of the topology named user2.mux3 into a file.

```
nt_shell> write_global_topology_submission \
          [get_lib_topology user2.mux3]
```

SEE ALSO

get_lib_topology(2)

```
list_topology_library(2)
report_topology_library(2)
```

write_parasitics

Writes parasitic data annotated on the design into a new file.

SYNTAX

```
status write_parasitics
   -format SPEF
   file_name
```

Data Types

```
   file_name            string
```

ARGUMENTS

-format SPEF

Specifies the format of the output parasitics file to be SPEF (Standard Parasitic Exchange Format).

file_name

The name of the parasitic data file written by the command.

DESCRIPTION

The **write_parasitics** command writes all parasitic data annotated on the current design to the file specified in the *file_name* argument. The **-format SPEF** option is required and specifies the format of the output file to be SPEF (Standard Parasitic Exchange Format).

COMMAND PHASING

The **write_parasitics** command can be executed any time after the **link_design** command.

EXAMPLES

The following example writes a parasitic data file in SPEF format.

```
nt_shell> write_parasitics -format SPEF top.spef
```

SEE ALSO

```
read_parasitics(2)
report_annotated_parasitics(2)
```

write_spice

Creates SPICE format files for timing analysis, noise analysis, or debugging.

SYNTAX

```
status write_spice
   [-header header_file_name]
   [-output file_name]
   [-use_wrapper_subckt]
   [-insert_model_cards]
   [-soi_comments]
   [-no_si_aggressors]
   [-si_noise above_high | below_high | above_low | below_low]
   [-si_noise_shape]
   [-fanout]
   [-fanout_net net_name]
   [-measure_subckt_delays]
   [-measure_subckt_transitions]
   [-dcs]
   [-pocv]
   [-simulate]
   objects
```

Data Types

```
   header_file_name        string
   file_name               string
   net_name                string
   objects                 list
```

ARGUMENTS

-header *header_file_name*

Specifies the path to a user-defined header file. The contents of the file are copied to the generated SPICE deck. You can use this file to identify the SPICE deck, to include the library files, to overwrite the default HSPICE simulation options, or to copy text to the SPICE deck for any other purposes.

-output *file_name*

Specifies the name of the SPICE deck file to be written. If this option is not used, the SPICE deck is written to the terminal screen.

-use_wrapper_subckt

If transistors are enclosed within wrapper subckts in the netlist, then this option causes the **write_spice** command to write the deck in terms of instances of wrapper subckts, instead of MOS devices, where possible. Use the **sim_transistor_wrapper_subckts** variable to specify which subckts are wrapper subckts if automatic identification is not correct. This option cannot be used with the **-insert_model_cards** option. This option is required if encrypted macro models are used.

-insert_model_cards

Causes the required model cards to be inserted into the deck. If the model cards were defined in .subckt blocks in the input SPICE deck, the model names used in the deck are machine-generated. This option cannot be used with the **-use_wrapper_subckt** option or with encrypted device models.

-soi_comments

Causes comments to be inserted in the SPICE deck for SOI transistors.

-no_si_aggressors

Causes the generated SPICE deck to exclude any cross-coupling capacitors to nets within the scope of the specified objects.

-si_noise

Specifies a noise analysis type or a list of noise analysis types. Valid values are **above_high**, **below_high**, **above_low** or **below_low**. If you specify more than one type, the tool generates multiple files, one file per noise type. This option causes the generated SPICE deck to include any cross-coupling capacitors to nets within the scope of the specified objects, along with a voltage driver on the aggressor side of each capacitor to represent the aggressor waveform. The victim is held high or low depend on the noise analysis type. The noise type also determines the aggressor switching direction.

-si_noise_shape

Causes the generated SPICE deck to include noise width and time-to-peak ratio measurements as well as user-defined input noise, if there is any.

-fanout

Causes the generated SPICE deck to include the fanout stage in the noise deck. The stage with the worst case fanout noise for the given noise type is chosen if **-fanout_net** option is not specified. The **-si_noise** option is mandatory with the **-fanout** option.

-fanout_net ***net_name***

Causes the generated SPICE deck to include the fanout stage whose output is the specified fanout net name. The **-si_noise** and **-fanout** options are mandatory with the **-fanout_net** option.

-measure_subckt_delays

Causes the generated SPICE deck to include HSPICE measurement statements that measure the delay across each subckt. Each subckt represents one or more arcs from the timing path.

-measure_subckt_transitions

Causes the generated SPICE deck to include HSPICE measurement statements that measure the transition time of each subckt output. Each subckt represents one or more arcs from the timing path.

-dcs

Creates a SPICE deck for the Dynamic Clock Simulation region. The header file and measurement options are taken from the standard DCS setup. This option is compatible only with the **-output**, **-simulate**, **-use_wrapper_subckt**, and **-insert_model_cards** options.

-pocv

Includes Monte Carlo directives in the SPICE deck to simulate parametric on-chip variation.

-simulate

Simulates the SPICE decks. The **-output** option is mandatory for the **-simulate** option to take effect. The simulation command, preprocessing command, and postprocessing command are defined by the **write_spice_sim_cmd**, **write_spice_sim_pre_processing**, and **write_spice_sim_post_processing** variables, respectively. Simulations are launched in parallel if distributed processing is enabled in the computing environment; otherwise, they are executed sequentially.

objects

Specifies a list of timing path objects, path arc objects, or net objects for which to write the SPICE files, one file per timing path, or one file per net per noise type. This option cannot be used with the **-dcs** option.

DESCRIPTION

The **write_spice** command writes out a SPICE file (or writes to the screen, if an output file is not specified) for the following purposes:

- To run dynamic simulation on a timing path or arc to obtain timing delays and input and output transition times
- To run dynamic simulation on a collection of nets for noise analysis
- To assist with debugging a dynamic clock simulation (DCS) region

Timing paths come from the **get_timing_paths** command, and path arcs come from the **arcs** attribute of a timing path. If you use the **-output** option and specify multiple timing paths, NanoTime generates a separate file for each timing path, using a different name for each file.

The SPICE deck is structured as a sequence of one-terminal and two-terminal subcircuits. Each subcircuit corresponds to one or more arcs in the timing path. Side inputs are sensitized locally in each subcircuit.

The generated SPICE deck includes a piecewise linear voltage source to drive the path and a transient analysis statement to control the SPICE run.

For paths that include dynamic delay simulations, the side inputs from the selected vector are represented as either DC voltage sources or voltage controlled voltage sources. If the vector includes delays between switching inputs, positive skews are represented by voltage sources with a time delay, whereas negative skews are converted to zero skew and a warning message is included in the SPICE deck.

SPICE deck files for timing paths or noise analysis include the results of the appropriate reporting command in the file header, written as comment lines. For a path delay SPICE deck, the report comes from the **report_paths** command. For a noise analysis SPICE deck, the report comes from the **report_noise** command. For a fanout noise analysis SPICE deck, the report comes from the **report_fanout_noise** command.

The default HSPICE simulation options are as follows:

```
\t.save level=none
\t.option lis_new=1 statfl=1 post=0 runlvl=6 autostop
```

You can override these options by including the new values in a header file and referencing the header file with the **-header** option.

The **-dcs** option is provided only to assist with debugging DCS regions.

COMMAND PHASING

If you want to use the SPICE deck for dynamic simulation, you must execute the **write_spice** command after path tracing is complete (in other words, after execution of the **trace_paths**, **extract_model**, or **characterize_context** command.

If you use the **-dcs** option to create a SPICE deck for debugging a dynamic clock simulation region, you can use the **write_spice** command any time after the **check_design** command.

EXAMPLES

The following example uses the **get_timing_paths** command to get a specific path, then writes a SPICE deck file named path.spc for the path.

```
nt_shell> write_spice -output path.spc \
          [get_timing_paths -max -rise_from d3 -rise_to q3]
```

The following example specifies to simulate the SPICE deck files that include head.sp as the header file and measure the delays and transition time of the subcircuits.

```
nt_shell> write_spice -simulate -output decks/deck.sp \
          -header head.sp -measure_subckt_delays \
          -measure_subckt_transition [get_timing_paths -max]
```

SEE ALSO

```
get_timing_paths(2)
sim_transistor_wrapper_subckts(3)
write_spice_sim_cmd(3)
write_spice_sim_max(3)
write_spice_sim_max_decks_per_task(3)
write_spice_sim_pre_processing(3)
```

```
write_spice_sim_post_processing(3)
distributed_processing_host_file(3)
distributed_processing_number_hspice_license(3)
```