PrimeTime Variables

Version C-2009.06, June 2009

SYNOPSYS®

Table of Contents

arch

This is a synonym for the read-only **sh_arch** variable.

SEE ALSO

sh_arch (3).

auto_link_disable

Disables the auto-link process.

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

When false (the default), many PrimeTime commands automatically attempt to link the current design for you; for example, **set_load** invokes the linker if the current design is not linked. Automatic linking occurs only if the design is completely unlinked. If the current design is partially linked and has unresolved references, automatic linking does not occur. If the current design is totally linked, there is no need for an auto-link, so it is not attempted.

Setting **auto_link_disable** to true disables the auto-link process. You can use this setting, along with the **link_design** command, to achieve the best possible performance when you have a large script that contains thousands of commands. Follow these steps:

1. Link the design manually with the **link_design** command.

2. Set **auto_link_disable** to true.

3. Source the script.

4. Reset **auto_link_disable** to false.

Note that setting the **auto_link_disable** variable to true is intended to be used in conjunction with a manual link step.

To determine the current value of this variable, type **printvar auto_link_disable** or **echo $auto_link_disable**.

SEE ALSO

link_design (2).

auto_wire_load_selection

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When *true* (the default), enables the automatic selection of wire load models, used to estimate net capacitances and resistances from the net fanout. When **false**, automatic selection of the wire load model is disabled.

The wire load models are described in the technology library. With the automatic selection of the wire load model, if the wire load mode is **segmented** or **enclosed**, the wire load model is chosen based on the area of the block containing the net either partially (for **segmented**) or fully (for **enclosed**). If the wire load mode is **top**, the wire load model is chosen based on the area of the top level design for all nets in the design hierarchy.

When you manually select a wire load model for a block (with the **set_wire_load_model** command), automatic wire load selection for that block is disabled.

To determine the current value of this variable, use **printvar auto_wire_load_selection**.

SEE ALSO

printvar (2), **report_wire_load** (2), **set_wire_load_min_block_size** (2).
set_wire_load_selection_group (2).

bus_naming_style

TYPE

fIstringfP

DEFAULT

DESCRIPTION

The **bus_naming_style** variable is used by the native Verilog reader to set the naming format for a specific element of a bus. This is the way that the names of the individual bits of the bus will appear in the application.

The default value of **bus_naming_style** is "%s[%d]". So, for example, for bus A, index 12, the name would be A[12].

To determine the current value of this variable, type **printvar bus_naming_style** or **echo $bus_naming_style**.

SEE ALSO

printvar (2), **read_verilog** (2).

case_analysis_log_file

Specifies the name of the file into which the details of case analysis propagation are written.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

Specifies the name of a log file to be generated during propagation of constant values from case analysis or from nets tied to logic zero or to logic one. The log file contains the list of all nets and pins that propagate constants. The constant propagation algorithm is an iterative process that propagates constants through nets and cells starting from a list of constant pins. The algorithm finishes when no more constants can be propagated. The format of the log file follows the constant propagation algorithm. For each iteration of the propagation process, the log file lists all nets and cells that propagate constants.

By default, this variable is set to an empty string, and no log file is generated during constant propagation.

To determine the current value of this variable, use **printvar case_analysis_log_file**.

SEE ALSO

remove_case_analysis (2), **report_case_analysis** (2), **report_disable_timing** (2), **set_case_analysis** (2); **disable_case_analysis** (3).

case_analysis_propagate_through_icg

Determines whether case analysis is propagated through integrated clock gating cells.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *false* (the default), constants propagating throughout the design will stop propagating when an integrated clock gating cell is encountered. Regardless of whether the integrated clock gating cell is enabled or disabled, no logic values will propagate in the fanout of the cell.

When *true*, constants propagated throughout the design will propagate through an integrated clock gating cell provided the cell is enabled. An integrated clock gating cell is enabled when its enable pin (or test enable pin) is set to a hi logic value. If the cell is disabled, then the disable logic value for the cell is propagated in its fanout. e.g. for a latch_posedge ICG, when it is disabled, it will propagate a logic 0 in its fanout.

To activate logic propagation through all integrated clock gating cells, the user must set the following prior to performing an update_timing.

```
set case_analysis_propagate_through_icg true
```

To determine the current value of this variable, type **printvar case_analysis_propagate_through_icg** or **echo $case_analysis_propagate_through_icg**.

SEE ALSO

set_case_analysis (2). **remove_case_analysis** (2).

case_analysis_sequential_propagation

Determines whether case analysis is propagated across sequential cells.

TYPE

fIstringfP

DEFAULT

never

DESCRIPTION

Determines whether case analysis is propagated across sequential cells. Allowed values are *never* (the default) or *always*. When set to *never*, case analysis is not propagated across the sequential cells. When set to *always*, case analysis is propagated across the sequential cells.

The one exception to sequential propagation occurs when dealing with sequential integrated clock gating cells. These types of ICG cells will only propagate logic values when the **case_analsyis_propagte_through_icg** variable is set to *true*.

To determine the current value of this variable, type **printvar case_analysis_sequential_propagation** or **echo $case_analysis_sequential_propagation**.

SEE ALSO

case_analysis_propagate_through_icg (3), **printvar** (2), **set_case_analysis** (2).

collection_deletion_effort

TYPE

fIstringfP

DEFAULT

low

DESCRIPTION

The **collection_deletion_effort** variable controls how saved collections are deleted when objects within them are potentially going out of scope. Allowed values are *low*, *medium*, or *high*, indicating to PrimeTime how much effort to expend to **preserve** a collection (or part of it) when objects are going out of scope.

Objects in a collection can go out of scope at several different times. When a cell is swapped or when the design is unlinked (that is, when another design is linked, causing the current linked design to become unlinked), a subset of objects in the design are removed, and collections can be affected. When the design/library which owns the objects is deleted, the collection is always deleted; **collection_deletion_effort** has no effect.

A collection is created relative to the current instance. That hierarchical node becomes the *root* of the collection. When hierarchical nodes that are being removed are above the root of a collection, the collection is always deleted. However, when the root of a collection is in the parent chain of the hierarchical node that is being deleted (henceforth referred to as the swap node), the collection is deleted based on the value of **collection_deletion_effort**:

• If the effort is *low*, the collection is deleted.

• If the effort is *medium*, the collection is deleted if **any** element of the collection has the swap node in its parent chain.

• If the effort is *high*, individual elements of the collection with the swap node in their parent chain are removed from the collection. The collection is deleted if it becomes empty.
The CPU cost increases from *low* to *high*. In most cases, *low* is a satisfactory choice.
To determine the current value of this variable, type **printvar collection_deletion_effort** or **echo $collection_deletion_effort**.

EXAMPLES

The following example illustrates the effects of using *low*, *medium*, or *high*.

In design ’M’, two nodes i1 and i2 reference design ’I’. The following collections are created:

```
pt_shell> set s1 [get_cells {i* i1/* i2/*}]
_sel27
pt_shell> query_objects $s1
{"i1", "i2", "i1/low", "i1/low2", "i1/low3", "i2/low", "i2/low2", "i2/low3"}
pt_shell> current_instance i1
i1
pt_shell> set s2 [get_cells *]
_sel28
pt_shell> query_objects $s2
{"i1/low", "i1/low2", "i1/low3"}
pt_shell> current_instance
Current instance is the top-level of design 'M'.
```

With **collection_deletion_effort** at *low*, you can swap i2/low with no effect on collection _sel28, stored in variable s2, because the root was i1. However, the collection _sel27, stored in variable s1, is deleted because its root (the top of design M) is in the parent chain of the swap node (i2/low).
With collection_deletion_effort at *medium*, you can swap i2/low with no effect on either collection _sel27 or _sel28. Here, the swap node i2/low is not above any element of either collection.
With collection_deletion_effort at *high*, you can swap i2 with no effect on collection _sel28. For collection _sel27, the elements in the collection with i2 in their parent chain are removed, leaving the following:

```
pt_shell> query_objects $s1
{"i1", "i2", "i1/low", "i1/low2", "i1/low3"}
```

SEE ALSO

collections(2), link_design(2), printvar(2), swap_cell(2).

collection_result_display_limit

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

TYPE

int

DEFAULT

100

DESCRIPTION

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

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

To determine the current value of this variable, use **printvar collection_result_display_limit**.

SEE ALSO

collections (2), **printvar** (2), **query_objects** (2).

create_clock_no_input_delay

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Affects delay propagation characteristics of clock sources created using **create_clock**. When *false* (the default), clock sources used in the data path are established as timing startpoints. The clock sources in the design propagate rising delays on every rising clock edge, and propagate falling delays on every falling clock edge. Disable this behavior by setting **create_clock_no_input_delay** to *true*.

To determine the current value of this variable, use **printvar create_clock_no_input_delay**.

SEE ALSO

create_clock(2).

dbr_ignore_external_links

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

When *false* (the default), if **read_db** encounters an external link when reading a DB file, it extracts as much information as possible from the DB file so that the linker can restore the link. When *true*, **read_db** ignores external links and searches for an object by name only in the libraries in the **link_path**.

External links are written by Design Compiler for objects (for example, wire load models and operating conditions), when there is a link from a design to another object in a library. The external link records information about the library to which the wire load was linked.

For example, if design TOP has an external link for a wire load model named "B100" in library "nominal.db", by default the linker attempts to load a library named "nominal.db", if it is not already loaded, then looks in that library for a wire load model named "B100". To override this default behavior and instead use "B100" from "min.db", you set **dbr_ignore_external_links** to *true*, and put "min.db" in the **link_path**.

To determine the current value of this variable, type **printvar dbr_ignore_external_links** or **echo $dbr_ignore_external_links**.

SEE ALSO

link_design (2), **printvar** (2), **read_db** (2); **link_path** (3), **search_path** (3).

default_oc_per_lib

Enables the use of a default operating condition per individual library.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Enables the use of a default operating condition per individual library. When the **default_oc_per_lib** variable is set to *true* (the default value), each cell that does not have an explicitly-set operating condition (on the cell itself, on any of its parent cells, or on the design) is assigned the default operating condition of the library to which the cell belongs. When set to *false* all cells that do not have any explicitly-set operating condition are assigned the default operating condition of the main library (the first library in the link_path).

The recommended flow is to explicitly set operating conditions on the design or on each hierarchical block that is powered by the same voltage (also called the voltage island). This variable is mainly for obtaining backward compatibility for the corner case of use of default conditions in releases prior to 2002.09.

To determine the current value of this variable, use **printvar default_oc_per_lib**.

SEE ALSO

set_operating_conditions (2); **link_path** (3).

disable_case_analysis

Specifies whether case analysis is disabled.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *false* (the default), constant propagation is performed in the design from pins either that are tied to a logic constant value, or for which a **case_analysis** command is specified. For example, a typical design has several pins set to a constant logic value. By default, this constant value propagates through the logic to which it connects. When the variable **disable_case_analysis** is *true*, case analysis and constant propagation are not performed.

To determine the current value of this variable, use **printvar disable_case_analysis**.

If the variable **disable_case_analysis_ti_hi_lo** is set to *true* then constant propagation from pins that are tied to a logic constant value will not be performed.

SEE ALSO

disable_case_analysis_ti_hi_lo (3), **remove_case_analysis** (2), **report_case_analysis** (2), **set_case_analysis** (2).

disable_case_analysis_ti_hi_lo

Specifies if logic constants should be propagated from pins that are tied to a logic constant value.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *false* (the default), constant propagation is performed from pins that are tied to a logic constant value.

For example, a typical design has several pins set to a constant logic value. By default, this constant value propagates through the logic to which it connects. When the variable **disable_case_analysis_ti_hi_lo** is *true*, constant propagation is not performed from these pins.

This current value of this variable does not alter the propagation of logic values from pins where the logic value has been set by the **set_case_analsyis** command.

To determine the current value of this variable, use **printvar disable_case_analysis_ti_hi_lo**.

If the variable **disable_case_analysis** is set to *true* then all constant propagation is disabled regardless of the current value of the **disable_case_analysis_ti_hi_lo** variable.

SEE ALSO

disable_case_analysis (3), **remove_case_analysis** (2), **report_case_analysis** (2), **set_case_analysis** (2).

eco_alternative_area_ratio_threshold

Specifies the maximum allowable area increase of a cell resize operation during the fixing process.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the maximum allowable area increase of a cell resize operation during the fixing process. The default value of zero (0) means that no limit is imposed on upsizing. However, if there is a specific requirement on the maximum size increase of a resized cell, you can control the area increase by setting this variable to a positive value which specifies the maximum upsize area increase as a ratio. By setting this variable to 2, upsizes would be limited to twice the area of the original cell. Imposing an upsize area limit can increase the timing predictability after physical implementation of the engineering change order (ECO) changes, as the smaller cell size increases are less likely to perturb layout during incremental placement and routing. However, more changes (and therefore more runtime) might be required to achieve the needed slack improvement.

SEE ALSO

fix_eco_timing(2)

eco_instance_name_prefix

Specifies the prefix used in generating insert_buffer cell instance names.

TYPE

string

DEFAULT

U

DESCRIPTION

This variable specifies the instance name prefix to be used by the insert_buffer command when creating new buffer/inverter cells. An instance number is appended to this prefix to form the new cell name. If the resulting cell name is already used, the instance number is incremented until an unused instance name is found. The default value of this variable is {U}, which causes insert_buffer to create instances U1, U2, U3, etc.

The rules for eco_instance_name_prefix are:

- **The first character must be A->Z or a->z**
- **Subsequent characters must be A->Z, a->z, 0->9, or _**

For example, consider a case where ECO changes are being made during the second iteration of timing closure. This variable can be set so that newly-created buffer and inverter instances are named in an easily-identifiable manner:

```
pt_shell> set eco_instance_name_prefix {Ueco2_}
Ueco2_
pt_shell> insert_buffer U2/Y slow/BUFX2
Information: Inserted 'Ueco2_1' at 'U2/Y'. (NED-046)
{"Ueco2_1"}
pt_shell> insert_buffer U2319/A slow/BUFX2
Information: Inserted 'Ueco2_2' at 'U2/A'. (NED-046)
{"Ueco2_2"}
pt_shell>
```

Changing the value of this variable does not affect the current value of the incrementer.

To determine the current value of this variable, use

```
pt_shell> printvar eco_instance_name_prefix
or
pt_shell> echo $eco_instance_name_prefix
```

SEE ALSO

insert_buffer (2),
eco_net_name_prefix (3).

eco_net_name_prefix

Specifies the prefix used in generating insert_buffer net names.

TYPE

string

DEFAULT

net

DESCRIPTION

This variable specifies the net name prefix to be used by the insert_buffer command when creating new nets. A net number is appended to this prefix to form the new net name. If the resulting net name is already used, the net number is incremented until an unused net name is found. The default value of this variable is {net}, which causes insert_buffer to create nets net1, net2, net3, etc.

The rules for eco_net_name_prefix are:

- **The first character must be A->Z or a->z**
- **Subsequent characters must be A->Z, a->z, 0->9, or _**

For example, consider a case where ECO changes are being made during the second iteration of timing closure. This variable can be set so that newly-created buffer and inverter nets are named in an easily-identifiable manner:

```
pt_shell> set eco_net_name_prefix {NETeco2_}
NETeco2_
pt_shell> insert_buffer U2/Y slow/BUFX2
Information: Inserted ’U1’ at ’U2/Y’. (NED-046)
{"U1"}
pt_shell> all_connected [get_pins U1/Z]
{"NETeco2_1"}
pt_shell>
```

Changing the value of this variable does not affect the current value of the incrementer.

To determine the current value of this variable, use

```
pt_shell> printvar eco_net_name_prefix
or
pt_shell> echo $eco_net_name_prefix
```

SEE ALSO

insert_buffer (2),
eco_instance_name_prefix (3).

eco_write_changes_prepend_libfile_to_libcell

Prepend link library file name information to library cell references in the **write_changes** change list.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When set to *false* (the default), references to library cells in the **write_changes** output contains only the library name and reference cell name information:

```
      pt_shell> write_changes
      create_cell {U1} {slow/BUFX1}
      insert_buffer [get_pins {U2320/Y}] slow/BUFX4 -new_net_names {net1} -
new_cell_names {U2}
      size_cell {U2} {slow/BUFX2}
```

When set to *true*, the library's file name information is prepended to the library name to fully specify the library cell reference:

```
      pt_shell> write_changes
      create_cell {U1} {slow.db:slow/BUFX1}
      insert_buffer [get_pins {U2320/Y}] slow.db:slow/BUFX4 -new_net_names {net1} -
new_cell_names {U2}
      size_cell {U2} {slow.db:slow/BUFX2}
```

This can be useful in multi-library flows, such as voltage islands, to fully specify the library that should be used to link the new instances. Only the library file name itself is prepended; the full file system path to the library is not included.

Note that this variable controls whether the library file name information is saved at the time of the ECO operation. Changing the value of the variable does not modify the **create_cell**, **insert_buffer** or **size_cell** operations, which have already been executed and recorded.

Note that variable has no effect if the **eco_write_changes_prepend_libname_to_libcell** variable is set to *false*. In this case, neither library name nor library file name are prepended.

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

pt_shell> **printvar eco_write_changes_prepend_libfile_to_libcell**

SEE ALSO

```
create_cell(3)
insert_buffer(3)
size_cell(3)
eco_write_changes_prepend_libname_to_libcell(3)
```

eco_write_changes_prepend_libname_to_libcell

Prepend link library information to library cell references in the **write_changes** change list.

TYPE

boolean

DEFAULT

true

DESCRIPTION

When set to *true* (the default), references to library cells in the **write_changes** output contain library name and reference cell name information:

```
      pt_shell> write_changes
      create_cell {U1} {slow/BUFX1}
      insert_buffer [get_pins {U2320/Y}] slow/BUFX4 -new_net_names {net1} -
new_cell_names {U2}
      size_cell {U2} {slow/BUFX2}
```

When set to *false*, the library's name is not prepended to the library name:

```
      pt_shell> write_changes
      create_cell {U1} {BUFX1}
      insert_buffer [get_pins {U2320/Y}] BUFX4 -new_net_names {net1} -
new_cell_names {U2}
      size_cell {U2} {BUFX2}
```

Note that the variable controls whether the library name information is saved at the time of the ECO operation. Changing the value of this variable does not modify **create_cell**, **insert_buffer** or **size_cell** operations that have already been executed and recorded.

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

pt_shell> **printvar eco_write_changes_prepend_libname_to_libcell**

SEE ALSO

create_cell(3)
insert_buffer(3)
size_cell(3)
eco_write_changes_prepend_libfile_to_libcell(3)

enable_license_auto_reduction

Determines whether or not the master returns licenses to the license server after a slave has finished using them.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When a slave finishs processing a task it returns the licenses it used to the master. When the *enable_license_auto_reduction* variable is *false* (the default), the master keeps the licenses checked out for future slave usage. When set to *true*, the master returns the licenses is receives from the slaves back to the license server unless another slave has already requested usage of that license.

The *enable_license_auto_reduction* variable should be enabled with care. When the master returns the licenses to the license server it may not be able to reacquire them. This will reduce the number of slaves that can execute concurrently leading to a degradation in performance during the subsequent analysis.

The *enable_license_auto_reduction* variable will only have an effect if the master was launched with incremental license handling enabled. Incremental license handling can be enabled by setting the UNIX environmental variable SNPSLMD_ENABLE_INCREMENTAL_LCIENSE_HANDLING to 1. See the documentation for further information.

To activate automatic license reduction, the user must set the variable as follows.

```
set enable_license_auto_reduction true
```

To determine the current value of this variable, type **printvar enable_license_auto_reduction** or **echo $enable_license_auto_reduction**.

enable_page_mode

This is a synonym for the **sh_enable_page_mode** variable.

DEFAULT

false

SEE ALSO

sh_enable_page_mode (3).

extract_model_capacitance_limit

Defines the maximum bound on the capacitance value for output ports of the netlist.

TYPE

float

DEFAULT

64.0

DESCRIPTION

Specifies the maximum permissible capacitance (load) at an output port. The **extract_model** command characterizes circuit timing from zero to the value specified by this variable. Setting a tight capacitance bound through this variable improves the accuracy of the extracted delay tables for a range of anticipated capacitive load. If a gate in the design does not have a **max_capacitance** attribute in its library definition, the **extract_model** command uses the value of the **extract_model_capacitance_limit** variable when adding a **max_capacitance** design rule attribute to output pins of the model. If the design rule is already defined in library, then this variable is used to establish a more constraining design rule in the resulting timing model. The default value of this variable is 64pf.

To determine the current value of this variable, type **printvar extract_model_capacitance_limit**.

SEE ALSO

extract_model (2), **extract_model_min_resolution**(3), **extract_model_data_transition_limit**(3). **extract_model_clock_transition_limit**(3). **extract_model_num_capacitance_points**(3).

extract_model_clock_transition_limit

Defines the maximum bound on the transition time (slew) for ports that transitively drive the CLK pins of registers.

TYPE

float

DEFAULT

5

DESCRIPTION

Defines the maximum bound on the transition time (slew) for ports that transitively drive the CLK pins of registers. Extracted timing tables are characterized for a transition range from zero to the specified bound. Specifying a tight bound for this variable improves the accuracy of extracted timing tables for a specified transition time range. The default value for this variable is 5.0ns. If a gate in the design whose input pin is connected to a clock port does not have a **max_transition** design rule in its library definition, the **extract_model** command uses the value of the **extract_model_data_transition_limit** variable when defining a **max_transition** design rule for the input port. If the design rule is already defined in library, then this variable is used to establish a more constraining design rule in the resulting timing model.

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

```
printvar extract_model_clock_transition_limit
```

SEE ALSO

extract_model (2), **extract_model_capacitance_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_capacitance_points** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3).

extract_model_data_transition_limit

Defines the maximum bound on the transition time (slew) for ports that transitively drive the D pins of registers.

TYPE

float

DEFAULT

5

DESCRIPTION

Defines the maximum bound on the transition time (slew) for ports that transitively drive the D pins of registers. Extracted timing tables are characterized for a transition range from zero to the specified bound. Specifying a tight bound for this variable improves the accuracy of extracted timing tables for a specified transition time range. The default value for this variable is 5.0. If a gate in the design whose input pin is connected to an input port does not have a **max_transition** design rule in its library definition, the **extract_model** command uses the value of the **extract_model_data_transition_limit** variable when defining a **max_transition** design rule for the input port. If the design rule is already defined in library, then this variable is used to establish a more constraining design rule in the resulting timing model.

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

```
printvar extract_model_data_transition_limit
```

SEE ALSO

extract_model (2), **extract_model_capacitance_limit** (3), **extract_model_clock_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_capacitance_points** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3).

extract_model_db_naming_compatibility

Determines whether the library name used in the DB format extracted model (ETM) should maintain the same naming style as before.

TYPE

string

DEFAULT

true

DESCRIPTION

ETM generated by PrimeTime can be written in either .lib (Liberty) or .db (Synopsys DB) format. Historically, the naming of the library in these 2 formats output can be deifferent. .lib format uses the string specified for the '-output' option; while the library naming of the .db format has been using the design name of the block being extracted. When the value is set to *true*, the above behavior will be unchanged for backward compatibility; if set to *falsefp, the .db format naming will follow the .lib format, i.e. using the value specified for the '-output' option. This may improve scripting convenience when both .lib and .db format ETM models are mixed in the flow.*

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

```
printvar extract_model_db_naming_compatibility
```

SEE ALSO

extract_model (2),

extract_model_enable_report_delay_calculation

Determines report delay calculation to be performed on the timing arcs contained in models.

TYPE

string

DEFAULT

true

DESCRIPTION

When the value is set to *false*, the models generated by PrimeTime model extraction do not allow report delay calculation to be performed on the timing arcs contained in models. The default value is *true*. This is enforceable only for the Synopsys database (.db) format output. For Liberty (.lib) format models, you can always modify the files before compiling them to enable or disable the feature of the resulting library.

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

```
printvar extract_model_enable_report_delay_calculation
```

SEE ALSO

extract_model (2), **report_delay_calculation** (2).

extract_model_gating_as_nochange

Controls the conversion from clock_gating setup/hold arcs into nochange arcs in the extracted model.

TYPE

string

DEFAULT

false

DESCRIPTION

When the value is set to *true*, clock gating setup and hold constraints are modeled as nochange arcs on the extracted model. When the value is set to *false*, clock gating checks are represented as separate setup and hold constraints. The default value is *false*. This variable affects models in all output formats: Synopsys database (.db) and Liberty (.lib) format.

For details and guidelines about the use of this variable and the method used to convert clock gating checks to nochange checks, see the *PrimeTime Modeling User Guide*.

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

```
printvar extract_model_gating_as_nochange
```

SEE ALSO

extract_model (2), **extract_model_capacitance_limit** (3), **extract_model_min_resolution** (3), **extract_model_tolerance** (3).

extract_model_include_ideal_clock_network_latency

Use this variable to control the behavior of accounting user defined network latencies for ideal clocks in the extracted timing models(ETM). By default, this variable is *false* and ETM does not account for any network latency for ideal clock trees, meaning delays to registers clocked by ideal clocks are always treated as 0.0 in the delay arcs.

TYPE

string

DEFAULT

false

DESCRIPTION

When set to *true*, indicates that PrimeTime model extraction will use the the user defined network latency for ideal clock paths leading to registers. This will impact the delay tables created for constraint arcs such as setup, hold from the ideal clock port to data input ports, as well as sequential rising/falling_edge delay arcs from ideal clock ports to output ports. It also impacts the clock insertion delay arcs created in the model if extraction of such arcs are enabled by **extract_model_with_clock_latency_arcs**.

When *false*, the default value, the extracted models will treat ideal clock paths as zero delay in creating timing arcs, the users are expected to re-apply the same clock network latency values when the model is used. This has been the same behavior of model extraction in and before 2004.06 PrimeTime releases.

The only modeling commands affected by this variable is **extract_model**. And all formats of the extraced models are affected.

To determine the current value of this variable, type **printvar extract_model_include_ideal_clock_network_latency**.

SEE ALSO

extract_model (2). **extract_model_with_clock_latency_arcs** (3).

extract_model_keep_inferred_nochange_arcs

Controls whether to keep PrimeTime inferred nochange relationships as nochange timing arcs in the extracted model.

TYPE

string

DEFAULT

false

DESCRIPTION

There are 2 ways for PrimeTime to perform a nochange constraint check between a data and a clock signal at a cell. The standard mechanism come explicitly from the library, when the timing arcs are defined as nochange timing types. The second mechanism is implicit. When interpreting cells based on its library arc types, PrimeTime can detect that between a data pin and a reference clock pin, there are pair-wise setup and hold arcs defined relative to the opposite edges of the clock signal, and the arcs do not form a standard edge-triggered regular flip-flop and also do not form a level-sensitive latch, PrimeTime may infer nochange relationship for the arc pair. For example, a setup_clock_rise and hold_clock_fall arc pair implies a nochange_clock_high check, meaning the data signal should be stable during the high pulse of the reference clock signal. This can be regarded as PrimeTime overrides the library defined arc types and treats the pair as forming a nochange type constraints instead of regular simple setup and hold.

In general, ETM gives precedence to the library defined timing types. This means, ETM always extracts those explicitly defined nochange arcs as nochange arcs in the model. For those implicitly inferred nochange arcs, by default, ETM extacts them as regular setup and hold arcs.

When the variable value is set to *true*, model extraction will align with PrimeTime timing analysis, detect those implicit nochange relationships and overrides the setup/hold arc types as nochange the same way as timing analysis does. In certain special path or arc configuration, this alignment provides better match between timing analysis with the original netlists and with the extracted model during model validation.

The default value is *false*.

This variable affects models in all output formats: Synopsys database (.db) and Liberty (.lib) format.

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

```
printvar extract_model_keep_inferred_nochange_arcs
```

SEE ALSO

extract_model (2)

extract_model_lib_format_with_check_pins

Determines if PrimeTime model extraction should write the internal check pins created for DB format models explicitly in the lib format model files.

TYPE

string

DEFAULT

false

DESCRIPTION

When the value is set to *true*, the lib format model files generated by PrimeTime model extraction will have the same set of internal check pins which are created in the DB format models under certain conditions in order for the PrimeTime timing engine to interpret the timing models correctly. The default value is *false*. This may mean that the check pins need to be created by the down stream tools that interpret the .lib model files.

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

```
printvar extract_model_lib_format_with_check_pins
```

SEE ALSO

extract_model (2), **report_delay_calculation** (2).

extract_model_merge_clock_gating

Indicates whether to merge clock gating setup and hold constraints with only nonclock gating clock constraints.

TYPE

string

DEFAULT

false

DESCRIPTION

When *true*, indicates that the PrimeTime **extract_model** command merges clock gating setup and hold constraints with non-clock gating clock constraints only keeping the most critical setup and hold constraints for any combination of input pin and clock edge. Even when the most critical constraint was originated from a clock gating constraint, the model constraint is not labeled as a clock gating constraint.

If you intend to use the generated models with a third party tool that does not accept multiple setup or hold constraint between the same input pin and clock edge, set this variable to *true*. When using models with Synopsys tools, set this variable to *false*.

To determine the current value of this variable, type **printvar extract_model_merge_clock_gating**.

SEE ALSO

extract_model (2);

extract_model_noise_iv_index_lower_factor

Controls the scale factor of the minimum index value used to create a steady-state current table (i.e., I-V curve).

TYPE

float

DEFAULT

-1.0

DESCRIPTION

Controls the scale factor of the minimum index value used to create a steady-state current table (i.e., I-V curve). This variable is a scale factor that is multiplied by Vdd (the power supply voltage). The default value is *-1.0*. For example, if Vdd = 1.8, then the minimum voltage used is -1.0*1.8 = -1.8.

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

```
printvar extract_model_noise_iv_index_lower_factor
```

SEE ALSO

extract_model (2), **report_noise** (2), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3), **extract_model_num_noise_width_points** (3), **extract_model_noise_iv_index_upper_factor** (3), **extract_model_num_noise_width_points** (3).

extract_model_noise_iv_index_upper_factor

Controls the scale factor of the minimum index value used to create a steady-state current table (i.e., I-V curve).

TYPE

float

DEFAULT

2.0

DESCRIPTION

Controls the scale factor of the maximum index value used to create a steady-state current table (i.e., I-V curve). This variable is a scale factor that is multiplied by Vdd (the power supply voltage). The default value is *2.0*. For example, if Vdd = 1.8, then the maximum voltage used is 2.0*1.8 = 3.6.

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

```
printvar extract_model_noise_iv_index_upper_factor
```

SEE ALSO

extract_model (2), **report_noise** (2), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3), **extract_model_num_noise_width_points** (3), **extract_model_noise_iv_index_lower_factor** (3), **extract_model_num_noise_width_points** (3).

extract_model_noise_width_points

Selects the exact noise width points of extracted noise immunity tables.

TYPE

string

DEFAULT

5

DESCRIPTION

Selects the exact noise width points of extracted noise immunity tables. The default value for this variable is an empty string. This value is only used if the -noise option is used with the extract_model command and the library's noise immunity tables will be selected by SI noise analysis for detecting noise on the input ports.

The value of this variable is a string of spaced float values.

Example: **set extract_model_noise_width_points "0.1 0.2 0.5 1.0"**

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

```
printvar extract_model_noise_width_points
```

SEE ALSO

extract_model (2), **report_noise** (2), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3). **extract_model_num_noise_width_points** (3),

extract_model_num_capacitance_points

Controls the size of extracted timing tables by defining the number of capacitance (load) points in these tables.

TYPE

int

DEFAULT

5

DESCRIPTION

Controls the size of extracted timing tables by defining the number of capacitance (load) points in these tables. The default value for this variable is *5*.

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

```
printvar extract_model_num_capacitance_points
```

SEE ALSO

extract_model (2), **extract_model_capacitance_limit** (3), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3).

extract_model_num_clock_transition_points

Controls the size of extracted timing tables by defining the number of clock transition time (slew) points in these tables.

TYPE

int

DEFAULT

5

DESCRIPTION

Controls the size of extracted timing tables by defining the number of clock transition time (slew) points in these tables. The default value for this variable is *5*.

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

printvar extract_model_num_clock_transition_points.

SEE ALSO

extract_model (2), **extract_model_capacitance_limit** (3), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_data_transition_points** (3), **extract_model_num_capacitance_points** (3).

extract_model_num_data_transition_points

Controls the size of extracted timing tables by defining the number of data transition time (slew) points in these tables.

TYPE

int

DEFAULT

5

DESCRIPTION

Controls the size of extracted timing tables by defining the number of data transition time (slew) points in these tables. The default value for this variable is 5.

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

printvar extract_model_num_data_transition_points.

SEE ALSO

extract_model (2), **extract_model_min_resolution** (3), **extract_model_capacitance_limit** (3), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_capacitance_points** (3).

extract_model_num_noise_iv_points

Controls the size of extracted noise steady-state current tables (i.e., I-V curve) by defining the number of index (i.e., voltage) points in these tables.

TYPE

int

DEFAULT

10

DESCRIPTION

Controls the size of extracted noise steady-state current tables (i.e., I-V curve) by defining the number of index (i.e., voltage) points in these tables. The default value for this variable is *10*. This value is only used if the -noise option is used with the extract_model command and if the library's steady-state current tables are selected by SI noise analysis for detecting noise on the output/inout ports.

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

```
printvar extract_model_num_noise_iv_points
```

SEE ALSO

extract_model (2), **report_noise** (2), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3), **extract_model_noise_iv_index_lower_factor** (3), **extract_model_noise_iv_index_upper_factor** (3), **extract_model_num_noise_width_points** (3).

extract_model_num_noise_width_points

Controls the size of extracted noise immunity tables by defining the number of noise width points in these tables.

TYPE

int

DEFAULT

5

DESCRIPTION

Controls the size of extracted noise immunity tables by defining the number of noise width points in these tables. The default value for this variable is *5*. This value is only used if the -noise option is used with the extract_model command and the library's noise immunity tables will be selected by SI noise analysis for detecting noise on the input ports.

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

```
printvar extract_model_num_noise_width_points
```

SEE ALSO

extract_model (2), **report_noise** (2), **extract_model_clock_transition_limit** (3), **extract_model_data_transition_limit** (3), **extract_model_min_resolution** (3), **extract_model_num_clock_transition_points** (3), **extract_model_num_data_transition_points** (3). **extract_model_noise_width_points** (3),

extract_model_single_pin_cap

This variable is to provide backward compatability with the introduction of min/max rise/fall specific pin capacitances in accounting for Miller effect. When "true"(the default value), it indicates that the models extracted will keep only a single capacitance value for a pin. The capacitance written is the maximum of all the min/ max rise/fall values.

TYPE

string

DEFAULT

true

DESCRIPTION

This variable is needed to deal with libraries accounting for Miller Effect by having different min/max and/or rise/fall capacitances for pins of library cells. The actual timing arcs extracted will not be impacted by this variable and still accounts for the Miller effect if it is available in the library and applicable in the analysis condition, meaning that for min/max paths, and rise/fall trasitions, the pin capacitances used to calculate the path delay are different if they are different in the library and the analysis type is not "single" operating condition.

When the variable is "false", and in .lib and DB formats, different min/max rise/ fall pin capacitance will be extracted if available and applicable, and written as per .lib and DB syntax.

The only modeling commands affected by this variable is the **extract_model**.

To determine the current value of this variable, type **printvar extract_model_single_pin_cap**.

SEE ALSO

extract_model (2).

extract_model_single_pin_cap_max

This variable is to provide a method to only write minimum capacitance values in a model that is in single capacitance mode. If the *extract_model_single_pin_cap* variable is true, the maximum capacitance is written to the model. If this variable is false, the minimum capacitance is written to the model.

TYPE

string

DEFAULT

true

DESCRIPTION

This variable is needed to deal with libraries accounting for Miller Effect by having different min/max and/or rise/fall capacitances for pins of library cells. The actual timing arcs extracted will not be impacted by this variable and still accounts for the Miller effect if it is available in the library and applicable in the analysis condition, meaning that for min/max paths, and rise/fall transitions, the pin capacitances used to calculate the path delay are different if they are different in the library and the analysis type is not "single" operating condition.

This variable only effects the model if the *extract_model_single_pin_cap* variable is set to *true*. If this variable is true and the minimum values if this variable is set to *false*, then the single capacitance written are the maximum values.

When the variable is *false* and in .lib and DB formats, different min/max rise/fall pin capacitance are extracted if available and applicable, and written as per .lib and DB syntax.

The only modeling commands affected by this variable is the **extract_model**.

To determine the current value of this variable, type **printvar extract_model_single_pin_cap_max**.

SEE ALSO

extract_model(2)
extract_model_single_pin_cap(3)

extract_model_split_partial_clock_gating_arcs

Controls whether to split the incomplete clock gating check setup/hold arcs in the extracted model.

TYPE

string

DEFAULT

false

DESCRIPTION

When the value is set to *true* and model extraction detects clock gating setup and hold constraints which cannot be paired together, a seperate internal check pin is created to avoid difference in timing analysis with the model in PrimeTime.

When the value is set to *false*, all clock gating checks are merged together and attached to the same pin. This is the default behavior.

This variable affects models in all output formats: Synopsys database (.db) and Liberty (.lib) format.

Standard clock gating constraint is essentially a no-change check to ensure that the active clock pulses are not clipped by the gating control signal. Therefore, setup and hold arcs need to be paired to check against opposite edges of the active clock pulse. For example, a setup on clock rise arc needs to be paired with a hold on clock fall edge to ensure no clipping of the positive clock pulse by the gating signal. However, there is no explicit syntax support in Library to define such arc pairing. The pairing thus has to be done automatically by PrimeTime when lirary cell timing arcs are analyzed. Consequently, this pair-wise relationship between setup and hold also determines the clock edges/cycles to be used when performing the checks. In particular, when setup check present, the hold check is performed on the edge whose opposit edege has been pre-chosen as the most constraining for setup analysis. In cases where a proper pairing relationship cannot be established among the arcs, PrimeTime treats the missing part as not being constrained. If the setup check is missing, hold is used as the primary constaint in choosing the most constaining edge.

As a compact timing model, ETM retains the most constraining relationship exists between an input port and its related clock port. There are many paths and end points that contribute to the constraints between them. ETM must worst-case and lump all the originally separate clock gating checks existed in pair but at different cells in the netlist into simple setup/hold arcs all between the same input and clock of the macro library cell. In doing so, we lose the details of the original arc/path pairing relationship, resulting in potentially different arc pairing when the ETM is used versus during netlist timing. Consequently, model validation may show a mismatch dur to differen clock edges are used for the orginally mis-pairing check. The difference arises most commonly when:

- The original setup/hold arcs in the library cells in the netlist are not standard clock-gating checks. For example, defined to be regarding the same clock edge.
- There are non-unate clock paths reaching the same cell, or mixture of inverting and non-inverting clock paths reaching different gating check cells.

Without the standard syntax support in Liberty to define arc pairing, to retain the original arc relationships, internal pins need to be introduced to force split of different classes of pairing on to different pins. This arc seperation controls the automatic pairing process when ETM is used in PrimeTime, thus reproducing the original timing behavior existed in the netlist.

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

printvar extract_model_split_partial_clock_gating_arcs

SEE ALSO

extract_model (2)
extract_model_capacitance_limit (3)

extract_model_status_level

Controls the message displaying for progress of the model extraction process.

TYPE

fIstringfP

DEFAULT

none

DESCRIPTION

Controls the number of progress messages displayed during the timing model extraction process. Allowed values are *none* (the default), *low*, *medium*, and *high*.

When set to *none*, no messages are displayed. When set to *low*, *medium*, or *high*, the progress of the model extraction is reported. The number of messages varies based on the value of the variable, as follows:

When set to *low*, messages are displayed at the beginning of major phases of the **extract_model** command.

When set to *medium*, all messages for *low* are displayed. Additionally, messages are displayed at the beginning of extraction subphases that may involve significant processing time.

When set to *high*, all messages for *medium* are displayed. Additionally, percent complete messages are displayed for the long running subphases.

To determine the current value of this variable, type **printvar extract_model_status_level** or **echo $extract_model_status_level**.

SEE ALSO

extract_model (2), **printvar** (2).

extract_model_suppress_three_state

Determines report delay calculation to be performed on the timing arcs contained in models.

TYPE

string

DEFAULT

false

DESCRIPTION

When the value is set to *true*, the models generated by PrimeTime model extraction will not contain the "is_three_state" pin attribute. The default value is *false*.

extract_model will add a "is_three_state" pin attribute to .lib models. The attribute is added for pins that are driven by three state logic. This allows for the extracted models to be used along with other three state drivers. PrimeTime will assume that only one of the three state drivers is driving at a time.

The "is_three_state" pin attribute is supported by LibraryCompiler versions greater than 2004.12-sp2, and by PrimeTime version greater than 2005.06.

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

```
printvar extract_model_suppress_three_state
```

SEE ALSO

extract_model (2)

extract_model_use_conservative_current_slew

Enables or disables the adjustment with the current context slew if it is more conservative during the extraction of a timing path. The adjusted models model the netlist behavior better if the netlist is in "worst_slew" propagation mode.

TYPE

string

DEFAULT

false

DESCRIPTION

When *false*, the default value, indicates no adjustment in the extraction and the models created will be the same as before. When set to *true*, PrimeTime model extractor adjusts the extracted tables with current context slew if it is more conservative to do so. This results in a model that will generally pass the model validation better if the netlist is also timed in "worst_slew" slew-propagation mode. The adjusted model is generally more pessimistic compare to the model extracted with the variable set to false.

This variable is only effective if the value of PrimeTime variable *timing_slew_propagation_mode* is set to "worst_slew" before doing the model extraction.

The only modeling commands affected by this variable is the **extract_model**. And all formats of the extraced models are affected.

To determine the current value of this variable, type **printvar extract_model_use_conservative_current_slew**.

SEE ALSO

extract_model (2), **timing_slew_propagation_mode** (3).

extract_model_with_3d_arcs

Enables or disables creating models contain arcs with 3-dimentional delay and transition tables.

TYPE

string

DEFAULT

true

DESCRIPTION

When *true*, the default value, indicates that PrimeTime model-generating commands try to merge certain output to output delay and/or clock to output arcs into 3D arcs with related_output_load as the third variable in the delay and/or transition tables. When set to *false*, the model keeps all output to output and/or clock to output constraint arcs as they are, which is the default behavior of model extraction in and before 2002.09 PrimeTime releases.

The only modeling commands affected by this variable is the **extract_model**. And all formats of the extraced models are affected. If your downstream tools do not know how to handle timing arcs with 3-dimensional delay tables, set the variable to *false* before extracting the timing model.

To determine the current value of this variable, type **printvar extract_model_with_3d_arcs**.

SEE ALSO

extract_model (2).

extract_model_with_clock_latency_arcs

Enables or disables creating clock tree latency, or clock insertion delay arcs in the extracted timing models(ETM).

TYPE

string

DEFAULT

false

DESCRIPTION

When set to *true*, indicates that PrimeTime model-generating commands will traverse all the clock tree paths, compute the insertion delay of the paths and create clock latency arcs in the model. Note clock insertion delay is the path delay measured between clock source and the destination registers, constraints from such as clock-gating cells are not considered. When *false*, the default value, the extracted models will not have clock insertion delay arcs in them. This is the default behavior of model extraction in and before 2003.03 PrimeTime releases.

The only modeling commands affected by this variable is **extract_model**. And all formats of the extraced models are affected.

If you extracted models with clock latency arcs in them and the format of the models is .lib, then you are required to use the 2003.12 or later LibraryCompiler to compile the model.

The clock latency arcs are meant to be used by tools such as Astro, PhysicalCompiler to compensate and balance clock tree skews at chip level. Those clock latency arcs in the models will also be respected by PrimeTime command **report_clock_timing** in reporting the clock tree latency and skews. Commands such as **report_delay_calculation**, **set_timing_derate**, **set_annotated_delay**, **get_timing_arcs**, etc. also recognize the new insertion delay arcs.

To determine the current value of this variable, type **printvar extract_model_with_clock_latency_arcs**.

SEE ALSO

extract_model (2). **report_clock_timing** (2).

extract_model_write_case_values_to_constraint_file

Controls whether to write logic values due to user set case analysis value propagation seperately into the ETM constraint file.

TYPE

string

DEFAULT

false

DESCRIPTION

By default, when the variable value is *false*, all logic constant values reaching block I/O ports are written into the .lib, .db files as "function" attribute for pin, regardless of whether the logic value is due to propagation of circuit intrinsic functional constants or user set case analysis values. This is the behavior of ETM.

When the value is set to *true*, and model extraction detects any logic constant set or propagated to the I/O boundary of the block as a result of user case analysis settings, the logic value is written into the ETM constraint file with proper **set_case_analysis** command.

This variable affects models in all output formats: Synopsys database (.db) and Liberty (.lib) format.

An ETM (Extracted Timing Model) generated by PrimeTime captures I/O timing behavior for static timing analysis (STA) purposes. By design, the primary flow intention is to improve the capacity and performance of downstream consumer tools in their timing driven implementation, optimization and analysis steps. Also by design, ETM does not retain any functional information of the original netlist and is essentially a functional "black-box". Lacking functional definition, ETM by itself may not be well suited for design steps where cell function is of primary concern. However, because logic values and their propagation do impact timing of both the block and higher level netlist where the model becomes instances, ETM has to keep the logic values from the block that propagated to the output ports, so that their effects to timing analysis can be carried consistently to higher level. These logic values can come from inherent netlist logic constants or from user set case analysis values. PrimeTime represents the logic values of the macro block with simple "function" attribute on pins. This ensures consistency from a timing analysis perspective. For certain design flows where some tools consuming ETM’s in certain steps need to be able to differentiate logic values resulted from functional constant vs case analysis propagation. We can optionally write logic values resulting from case analysis to a seperate constraint file, and only write the functional constant with "function" attribute.

It is worth noting that the constraint file written along with the .lib and/or .db model files should be used together with the ETM in order to completely reproduce the timing behavior of the original netlist with the model.

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

```
printvar extract_model_write_case_values_to_constraint_file
```

SEE ALSO

extract_model (2), **set_case_analysis** (2).

hier_scope_check_defaults

Defines the types of scope checks to be performed by the **check_block_scope** command, or reported by **report_scope_data** command. Note that this variable does not impact the scope information to be captured by PrimeTime model creation commands **create_ilm** and **extract_model**. The allowed value for this variable is one or more of the following: clock_arrival, clock_transition, data_input_arrival, data_input_transition, clock_uncertainty and clock_skew_with_uncertainty.

TYPE

list

DEFAULT

clock_arrival clock_transition clock_skew_with_uncertainty

GROUP

hierarcy_variables

DESCRIPTION

All the applicable scoping information is captured and store in files during the block-level analysis and model creation time. When integrating at the top-level with some blocks being replaced by their timing models, it is recommended that users perform the checks to confirm that the models are indeed instantiated within the ranges they are originally validated for. Any violations reported by the scope check for the model can potentially result in timing violations, but there is no implication by the scope violation on whether, where and how much the timing violations would be.

User can use this variable to customize the types of scope checks to be performed by the **check_block_scope** command at top-level and concentrate on the ones that are important for the flow and each instance.

SEE ALSO

check_block_scope (2). **report_scope_data** (2). **create_ilm** (2). **extract_model** (2).

hierarchy_separator

TYPE

fIstringfP

DEFAULT

/ (forward slash)

DESCRIPTION

Determines how hierarchical elements of the netlist are delimited in reports, and searched for in selections and other commands. By default, the value is the slash, "/". The choice of a separator is limited to these characters: bar "|", caret "^", at "@", dot ".", and slash "/".

Normally, you should accept the default slash. However, in some cases where the hierarchy character is embedded in some names, the search engine might produce results that are not intended; the **hierarchy_separator** is a convenient method for dealing with this situation. For example, 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 intended. However, if you set the **hierarchy_separator** to the vertical bar "|", searching for "A | B/C | D" is very explicit, as is "A | B | C | D".

SEE ALSO

selection (2).

ilm_ignore_percentage

Specifies a threshold for the percentage of total registers in the transitive fanout of an input port, beyond which the port is to be ignored when identifying interface logic.

TYPE

float

DEFAULT

25

DESCRIPTION

Specifies a minimum threshold for the percentage of the total registers in the transitive fanout of an input port, beyond which the port is to be ignored when identifying interface logic. The default is 25. This variable affects the **-auto_ignore** option of the **identify_interface_logic command**. **-auto_ignore** determines automatically those ports (for example, scan enable and reset ports) that should be ignored when **identify_interface_logic** places the **is_interface_logic_pin** attribute on objects to identify them as part of the interface logic model (ILM) for the design. Ports are automatically ignored if they fan out to a percentage of total registers in the design greater than the value specified by this variable.

For a discussion of ILM creation and the associated commands, see the manual page for the **identify_interface_logic** command.

To determine the current value of this variable, type **printvar ilm_ignore_percentage**.

SEE ALSO

identify_interface_logic (2).

ilm_write_verilog_logic_constant_net_names

Specifies that the logic constant net names should be written as 1’b0 or 1’b1 rather than the default way of writing them as nets *Logic0* and *Logic1* and setting a case analysis on corresponding pins.

TYPE

boolean

DEFAULT

false

DESCRIPTION

Specifies that the logic constant net names should be written as 1’b0 or 1’b1 in the ILM verilog. The default method is to create nets *Logic0* and *Logic1* and set a case analysis on the corresponding pins in ILM constraint file.

For a discussion of ILM creation and the associated commands, see the manual page for the **create_ilm** command.

To determine the current value of this variable, type **printvar ilm_write_verilog_logic_constant_net_names**.

SEE ALSO

create_ilm (2).

in_gui_session

This read-only variable is *true* when the GUI is active and *false* (the default) when the GUI is inactive.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

You can use this variable when writing Tcl code that depends on the presence of the graphical user interface (GUI). If you have invoked the **gui_start** command and the GUI is active, the read-only variable is *true*. Otherwise, the variable is *false* and the GUI is inactive.

SEE ALSO

printvar (2)
gui_start(2)
gui_stop (2)

lib_thresholds_per_lib

Enables waveform measurement thresholds and rail-voltages to be taken from each individual library.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **lib_thresholds_per_lib** variable is set to *true* (the default value), waveform measurement thresholds and rail-voltages are taken from each individual library; this is the most accurate setup. For backward compatibility with previous releases of PrimeTime, you can set this variable to *false*, in which case the thresholds are taken from the first library in the link_path (the "main" library) and rail-voltages are ignored for the purposes of setting thresholds.

To determine the current value of this variable, use **printvar lib_thresholds_per_lib**.

Subtleties: With old versions of Library Compiler it was possible to create libraries without thresholds. This necessitated the use of **rc_*_threshold_*** shell variables to convey the settings to PrimeTime. Recent versions of Library Compiler write out default values, if the thresholds are not set in the library source file. You can check the threshold settings for a particular library with the **report_lib** command. You can check the threshold settings used for a particular delay-calculation arc with the **report_delay_calculation -thresholds** command.

When **lib_thresholds_per_lib** is *true* (the default), PrimeTime looks for the thresholds first in the library containing the pin, then in the **rc_*_threshold_*** shell variables if set, and then in the main library. If none of these sources provide valid thresholds, PrimeTime will use 20%-80% slew and 50% delay thresholds as a fallback.

When **lib_thresholds_per_lib** is *false*, PrimeTime looks for the thresholds first in the **rc_*_threshold_*** shell variables if set and then in the main library. If none of these sources provide valid thresholds, PrimeTime will use 20%-80% slew and 50% delay thresholds as a fallback.

Thresholds are specified as a percentage of the rail-voltage, so that when **lib_thresholds_per_lib** is *true*, libraries with identical threshold percentages but different rail-voltages will have different threshold voltages. When **lib_thresholds_per_lib** is *false*, the main library thresholds are used as percentages of the main library rail-voltage for all pins, so rail-voltages are ignored for the purposes of setting thresholds. Note that rail-voltages for the purpose of delay calculation are used based on the set operating conditions. For full backward compatibility you may need to use **timing_prelayout_scaling** to disable prelayout delay scaling and **set_operating_conditions** to force single voltage on the design.

Delay calculation for a network arc can fail if the network drivers are incapable of generating a waveform that reaches all of a load’s threshold voltages. See the man page for RC-005 for more information.

SEE ALSO

report_delay_calculation (2), **report_lib** (2); **set_operating_conditions** (2); **timing_prelayout_scaling** (3), **rc_input_threshold_pct_fall** (3), **rc_input_threshold_pct_rise** (3), **rc_output_threshold_pct_fall** (3), **rc_output_threshold_pct_rise** (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), **rc_slew_derate_from_library** (3); **RC-005** (n).

link_create_black_boxes

Enables the linker to automatically convert each unresolved reference into a black box.

TYPE

fIBooleanfP

DEFAULT

true

DESCRIPTION

When this variable is set to *true* (the default), the linker automatically converts each unresolved reference into a black box, which is essentially an empty cell with no timing arcs. The result is a completely linked design on which analysis can be performed.

When you set this variable to *false*, unresolved references remain u nresolved and most analysis commands cannot function.

To determine the current value of this variable, type **printvar link_create_black_boxes** or **echo $link_create_black_boxes**

SEE ALSO

link_design (2)
printvar (2)
search_path (3)

link_force_case

Controls the case sensitivity behavior of the link design command.

TYPE

fIstringfP

DEFAULT

check_reference

DESCRIPTION

This variable controls the case-sensitive or insensitive behavior of the **link_design** command. Allowed values are *check_reference* (the default), *case_sensitive*, or *case_insensitive*. The *check_reference* option means that the case sensitivity of the link is determined only by the case sensitivity of the input format that created that reference. For example, a VHDL reference is linked case-insensitively, whereas a Verilog reference is linked case-sensitively.

Some caveats apply to this variable, as follows:

1. Do not set the **link_force_case** variable to *case_insensitive* if you are reading in source files from case-sensitive formats (for example, Verilog). Doing so could cause inconsistent, unexpected, and undesirable results.

For example, you might have an instance u1 of design 'inter', but might have loaded a design 'Inter'. If you do a case-insensitive link, you get design 'Inter'. The side effect is that the relationship between u1 and 'inter' is gone; it has been replaced by a relationship between u1 and 'Inter'. Changing the **link_force_case** variable back to *check_reference* or *case_sensitive* does not restore the original relationship. You would have to remove the top design, reload it, and relink. (Note that **dc_shell** has the same restriction.)

2. Do not change the value of this variable within a session. Doing so could cause numerous error and warning messages, which could be confusing.

3. Be aware that setting the **link_force_case** variable to *case_insensitive* can decrease the performance of the **link_design** command.

To determine the current value of this variable, type **printvar link_force_case** or **echo $link_force_case**

For a list of all link-related variables and their current values, use the **printvar link*** command.

SEE ALSO

link_design (2)
link_create_black_boxes (3)

link_library

This is a synonym for the **link_path** variable.

SEE ALSO

link_path (3).

link_path

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

TYPE

fIlistfP

DEFAULT

*

DESCRIPTION

Specifies a list of libraries, design files, and library files used during linking. The **link_design** command looks at those files and tries to resolve references in the order of the specified files.

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

The "*" entry in the value of this variable indicates that the **link_design** command should search all the designs loaded in **pt_shell** while trying to resolve references. Designs are searched in the order in which they were read.

For elements other than "*", **pt_shell** searches for a library that has already been loaded. If that search fails, **pt_shell** searches for a file name using the **search_path** variable.

The default value of the **link_path** variable is "*". To determine the current value of this variable, type **printvar link_path** or **echo $link_path**.

SEE ALSO

link_design (2)
printvar (2)
search_path (3)

link_path_per_instance

Overrides the default link path for selected leaf cell or hierarchical cell instances.

TYPE

fIlistfP

DEFAULT

(empty)

DESCRIPTION

This variable, which takes effect only if set prior to linking the current design, overrides the default link_path variable for selected leaf cell or hierarchical cell instances. The format is a list of lists. Each sublist consists of a pair of elements: a set of instances, and a **link_path** specification which should be used for and within these instances. For example,

```
set link_path {* lib1.db}
set link_path_per_instance [list
   [list {ucore} {* lib2.db}]
   [list {ucore/usubblk} {* lib3.db}]]
```

Entries are used to link the specified level and below. If a given block matches multiple entries in the per-instance list, the more specific entry overrides the more general entry. In the example above:

1. lib3.db would be used to link blocks ’ucore/usubblk’ and below.

2. lib2.db would be used to link ’ucore’ and below (except within ’ucore/subblk’).

3. lib1.db would be used for the remainder of the design (everything except within ’ucore’).

The default value of the **link_path_per_instance** variable is an empty list, meaning that the feature is disabled. To determine the current value of this variable, type **set link_path_per_instance** or **echo $link_path_per_instance**.

SEE ALSO

link_design (2)
link_path (3)

multi_core_enable_analysis

Enables or disables PrimeTime multicore analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting this variable to true enables the multicore analysis flow. Multicore analysis reduces the runtime of a PrimeTime session by parallelizing the timing update and a selection of reporting commands. This is achieved by partitioning the analysis problem and distributing these partitions for analysis on multiple processors, potentially among different machines on a network. By default, this variable is set to false and all commands are executed according to the single-core analysis flow.

To determine the current value of this variable, type

```
      pt_shell> printvar multi_core_enable_analysis
```

Altering the value of this variable schedules a full timing update. If, at the start of the distributed timing update, PrimeTime encounters features that are unsupported in the multicore analysis flow, the value of the **multi_core_enable_analysis** variable is automatically set to false and PrimeTime defaults to the single-core analysis flow with a warning message.

You can alter the setting of this variable at any stage in the flow.

Multicore analysis uses the concept of a host. A host specifies a unique process associated with a given compute server or single computational slot on a managed farm of compute servers, for example, a Load Sharing Facility (LSF) or Grid Engine (GRD). For a distributed analysis, you must specify the host options for compute resources. For more information, see the **set_host_options** and **start_hosts** commands.

Some reports and commands require the transmission of data from the remote processes to the master to transform it into a state equivalent to an up-to-date single-core analysis. Details of the commands that initiate this type of behavior can be found in the user guides. Quality of results (QoR) is not affected.

Multicore analysis is a single update feature by default. Any post distributed update change command causes PrimeTime to revert to single-core analysis, and to terminate the remote processes, meaning that the next update is purely single-core and any subsequent updates are performed incrementally. To re-enable distributed analysis, you can explicitly set the **multi_core_enable_analysis** variable back to

TRUE - the resulting distributed update will be a full update, requiring a re-partitioning, re- dispatch and delay calculation.

Save and restore functional capabilities are also supported in the multicore analysis flow. Restoring a session restores PrimeTime to a state equal to that at the point of saving. Thus, a single-core analysis PrimeTime session can restore a multicore analysis saved image, but by doing so it transforms to a multicore analysis PrimeTime session. Similarly, a multicore analysis session can restore a single-core analysis saved image by becoming a single-core analysis flow.

EXAMPLE

The following shows an example of the setup for multicore analysis in PrimeTime. In the example, you use the **set_hosts_options** command to specify two partition processes called "OPTS_1", which are using the same architecture as the master and are on the same host as the master.

```
      pt_shell> set multi_core_enable_analysis true
      true
      pt_shell> set_host_options -name OPTS_1 -num_processes 2
      1
      pt_shell> start_hosts
         1] Launched  : ...
                        ...
              Status  : Forking successful
              Stdout  : your job
              Stderr  : **<<EMPTY>>**
         2] Launched  : ...

         ...
         ...

        ----------------------------------------------------------------------------
------------
        Distributed farm creation timeout : 21600 seconds
        Waiting for   2 (of  2) distributed hosts (Wed Nov 19 07:29:35 2008)
        Waiting for   0 (of  2) distributed hosts (Wed Nov 19 07:30:05 2008)
        ----------------------------------------------------------------------------
------------

      1
      pt_shell> report_hosts -verbose
      ****************************************
      Report : hosts
              -verbose
      Version: B-2008.12
      Date   : Wed Nov 19 07:30:06 2008
      ****************************************

       Options         Host              32Bit Num       Submit            Submit
       Name            Name                    Processes Command           Options
       ------------------------------------------------------------------------------
--
```

```
         OPTS_1          **localhost**     N     2

         Options         Process                 Status
         Name
         --------------------------------------------------
         OPTS_1          1                       ONLINE
                         2                       ONLINE

        1
        pt_shell> update_timing

        1
```

SEE ALSO

printvar (2), **set_host_options** (2), **remove_host_options** (2), **report_host_options** (2), **start_hosts** (2), **remove_hosts** (2), **stop_hosts** (2).

multi_core_skip_unsupported

Enables or disables skipping of unsupported commands and variables in the multicore analysis flow.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls how unsupported commands and variables are handled by PrimeTime in the multicore analysis flow.

By default, this variable is set to *true* and all commands that are unsupported in multicore analysis are skipped and a warning is issued. When **multi_core_skip_unsupported** is set to *true* (the default), unsupported variables are reset to their default value and a warning issued to allow the multicore analysis to proceed.

Upon exiting the session, PrimeTime produces a summary of commands and variables that were skipped along with the location of a data file containing further information. The data file is located in the multicore analysis working directory, which you specified using the **multi_core_working_directory** variable.

To execute an unsupported command or use an unsupported variable, the **multi_core_skip_unsupported** should be set to *false*. If the **multi_core_skip_unsupported** variable is set to *false* and an unsupported command or variable is encountered, PrimeTime displays a warning message, automatically sets the value of the **multi_core_enable_analysis** variable to *false*, and returns to the single-core analysis flow.

You can alter the setting of this variable at any stage in the multicore analysis flow.

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

```
printvar multi_core_skip_unsupported
echo $multi_core_skip_unsupported
```

SEE ALSO

printvar(2)
echo(2)
multi_core_enable_analysis(3)
multi_core_working_directory(3)

multi_core_working_directory

Defines the root working directory for all multi-core analysis data, including remote process log files.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

In multicore analysis, the working directory may be explicity specified using the **multi_core_working_directory** variable. This defines the root working directory for all multi-core analysis data, including logfiles. The specified directory must be write-accessible and network accessible by the user and all PrimeTime processes. This value of this variable cannot be modified once the remote hosts have been launched. If no value is specified for **multi_core_working_directory**, the working directory will default to the directory from which the analysis was invoked. To determine the current value of this variable, type the following:

```
pt_shell> printvar multi_core_working_directory
```

or

pt_shell> echo $multi_core_working_directory.

SEE ALSO

printvar (2), **multi_core_enable_analysis** (3).

multi_scenario_merged_error_limit

Defines the maximum number of errors or warnings of a particular type to be considered for merging in the merged error log on a per task basis.

TYPE

int

DEFAULT

100

DESCRIPTION

In multi-scenario analysis, errors, warnings, and informational messages are produced by both the master and slave processes. For the data produced by the slaves, full (compressed) error, warning, and informational messages are stored in the file specified by the **multi_scenario_merged_error_log** variable. To avoid excessive memory usage and network traffic, the number of errors on a per type basis is limited to the number set using the **multi_scenario_merged_error_limit** variable. For example, if the **multi_scenario_merged_error_limit** variable is set to 10, a maximum of 10 errors of type CMD-003 is written to the log specified using the **multi_scenario_merged_error_log** variable. The default value of the **multi_scenario_merged_error_limit** is 100.

To determine the current value of this variable, type **printvar multi_scenario_merged_error_limit**

SEE ALSO

multi_scenario_merged_error_log(3)

multi_scenario_merged_error_log

Use to specify a file location where full (compressed) error, warning and informational messages are stored for data produced by the slaves. .

TYPE

string

DESCRIPTION

In multi-scenario analysis, errors, warnings and informational messages are produced by both the master and slave processes. For the data produced by the slaves, full (compressed) error, warning and informational messages are stored in the file specified by **multi_scenario_merged_error_log**. All errors, warnings and informationals are displayed as in regular PrimeTime, with the addition of the name of the scenario where the message occured. If all scenarios produce the same message, only one message is stored with all used instead of listing all scenario names. This describes the term compressed used above. Along with the multi-scenario merged error log, slave errors are displayed in full at the master while slave warnings are summarized the master (slave informationals are not displayed at the master) . At each slave, the errors and warnings are written in full to the slave log file.

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

or

echo $multi_scenario_merged_error_log.

SEE ALSO

printvar (2).

multi_scenario_message_verbosity_level

Define the verbosity level of messages printed at the master during slave processing.

TYPE

string

DEFAULT

default

DESCRIPTION

During slave processing, the master reports back several different types of messages. You can use the **multi_scenario_message_verbosity_level** variable to control the types of messages reported at the master. The variable can take two values, low or default:

```
low - The master prints out the following messages:
errors, fatal, and task failure

default - The master prints out the following messages:
licensing, errors, warnings, fatal, task status, and task failure
```

To determine the current value of this variable, type **printvar multi_scenario_message_verbosity_level**

SEE ALSO

multi_scenario_merged_error_log(3)

multi_scenario_working_directory

Defines the root working directory for all multi-scenario analysis data, including log files.

TYPE

string

DESCRIPTION

In multi-scenario analysis, the working directory may be explicity specified using the **multi_scenario_working_directory** variable. This defines the root working directory for all multi-scenario analysis data, including logfiles. The working directory must be visible to the master and all the slaves. The user must have write permissions in the specified directory. If no value is specified for **multi_scenario_working_directory**, the working directory will default to the directory from which the analysis was invoked. To determine the current value of this variable, type the following:

```
printvar multi_scenario_working_directory
```

or

echo $multi_scenario_working_directory.

SEE ALSO

printvar (2).

mw_design_library

This variable stores the name of the design library for the **read_milkyway** command.

TYPE

fIStringfP

DEFAULT

"" (empty)

GROUP

milkyway variables

DESCRIPTION

This variable has the name of the design library for the **read_milkyway** command. If the **read_milkyway** command is issued without the design library name on the command line, then it reads this variable to get the name. If the **read_milkyway** comamnd is issued with the design library name then this variable is set to that name.
To determine the current value of this variable, type **printvar mw_design_library** or **echo $mw_design_library**.

SEE ALSO

read_milkyway (2)

mw_logic0_net

This variable controls the name of constant zero net for the **read_milkyway** command.

TYPE

fIStringfP

DEFAULT

VSS

GROUP

milkyway variables

DESCRIPTION

This variable determines the name of the net is the logic low net for the **read_milkyway** command. When **read_milkyway** is reading a design and sees a net by this name it treats it as if it were tied to logic 0.
To determine the current value of this variable, type **printvar mw_logic0_net** or **echo $mw_logic0_net**.

SEE ALSO

read_milkyway (2), **mw_logic1_net** (3)

mw_logic1_net

This variable controls the name of the constant one net for the **read_milkyway** command.

TYPE

fIStringfP

DEFAULT

VDD

GROUP

milkyway variables

DESCRIPTION

This variable determines the name of the net is the logic high net for the **read_milkyway** command. When **read_milkyway** is reading a design and sees a net by this name it treats it as if it were tied to logic 1.
To determine the current value of this variable, type **printvar mw_logic1_net** or **echo $mw_logic1_net**.

SEE ALSO

read_milkyway (2), **mw_logic0_net** (3)

variation_pba_use_worst_parasitics

Specifies the type of default distribution used for parasitic variation parameters in PrimeTime-VX.

TYPE

String

DEFAULT

normal

DESCRIPTION

This variable is used to specify the default distribution of parasitic variation parameters, for statistical static timing analysis of the design. This variable is applicable only when variation aware parasitics are read using **read_parasitics -keep_variations -create_default** command.
The valid options are normal (the default), uniform and discrete.
When set to normal, a Gaussian distribution is created for the parasitic variation parameters. When set to uniform, a uniform distribution from mean-abs(3*sigma) to mean+abs(3*sigma) is created. When set to discrete, a 2-value distribution for the parasitic variation parameters is created. The two values used are mean+abs(3*sigma) and mean-abs(3*sigma), each with equal probability of 0.5. The distribution used for the parameters can be seen using **report_variation** command.
To determine the current value of this variable, type **printvar parasitic_variation_default_type** or **echo $parasitic_variation_default_type**.

SEE ALSO

read_parasitics(2), report_variation(2)

parasitics_cap_warning_threshold

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

TYPE

fIfloatfP

DEFAULT

0.0

DESCRIPTION

When this variable is set with a value greater than 0.0, **read_parasitics** issues a *PARA-014* warning if it finds in the parasitics file a capacitance value, in picofarads, greater than this threshold. The default is 0.0, in which case no checking is done. Use this variable to detect large, unexpected capacitance values written to parasitics files by other applications. The capacitor is still used, but you can quickly find it in the parasitics file.

You can define an analogous resistance threshold by using the **parasitics_res_warning_threshold** variable.

To determine the current value of this variable, type **printvar parasitics_cap_warning_threshold** or **echo $parasitics_cap_warning_threshold**.

SEE ALSO

read_parasitics (2). **parasitics_res_warning_threshold** (3). **PARA-014** (n).

parasitics_rejection_net_size

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

TYPE

int

DEFAULT

20000

DESCRIPTION

Defines a threshold number of nodes in an annotated parasitic network beyond which the detailed network is automatically replaced by a lumped capacitance. The default is 20000.

This variable is one of a pair of variables, **parasitics_warning_net_size** and **parasitics_rejection_net_size**, that help you prevent unacceptable or unexpected run times caused by large numbers of nodes in an annotated parasitic network. If the **read_parasitics** command finds a number of nodes that exceeds the value of **parasitics_warning_net_size** (default 10000), you receive a message (PARA-003) warning you that extended run time could occur. If the **read_parasitics** command finds a number of nodes that exceeds the value of **parasitics_rejection_net_size** (default 20000), the network is rejected and automatically replaced by a lumped capacitance. You receive a message (PARA-004) warning you of that action.

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

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

```
pt_shell> printvar parasitics_rejection_net_size
```

SEE ALSO

read_parasitics (2); **parasitics_warning_net_size** (3).

parasitics_res_warning_threshold

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

TYPE

fIfloatfP

DEFAULT

0.0

DESCRIPTION

When this variable is set with a value greater than 0.0, **read_parasitics** issues a *PARA-014* warning if it finds in the parasitics file a resistance value, in ohms, greater than this threshold. The default is 0.0, in which case no checking is done. Use this variable to detect large, unexpected resistance values written to parasitics files by other applications. The resistor is still used, but you can quickly find it in the parasitics file.

You can define an analogous capacitance threshold by using the **parasitics_cap_warning_threshold** variable.

To determine the current value of this variable, type **printvar parasitics_res_warning_threshold** or **echo $parasitics_res_warning_threshold**.

SEE ALSO

read_parasitics (2). **parasitics_cap_warning_threshold** (3). **PARA-014** (n).

parasitics_warning_net_size

Defines a threshold number of nodes in an annotated parasitic network beyond which a message is issued that warns of a potential extended run time.

TYPE

int

DEFAULT

10000

DESCRIPTION

Defines a threshold number of nodes in an annotated parasitic network beyond which a message is issued that warns of a potential extended run time. The default is 10000.

This variable is one of a pair of variables, **parasitics_warning_net_size** and **parasitics_rejection_net_size**, that help you prevent unacceptable or unexpected run times caused by large numbers of nodes in an annotated parasitic network. If the **read_parasitics** command finds a number of nodes that exceeds the value of **parasitics_warning_net_size** (default 10000), you receive a message (PARA-003) warning you that extended run time could occur. If the **read_parasitics** command finds a number of nodes that exceeds the value of **parasitics_rejection_net_size** (default 20000), the network is rejected and automatically replaced by a lumped capacitance. You receive a message (PARA-004) warning you of that action.

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

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

```
pt_shell> printvar parasitics_warning_net_size
```

SEE ALSO

read_parasitics (2); **parasitics_rejection_net_size(3).**

pba_aocvm_only_mode

Specifies that only AOCVM is performed during path-based analysis.

TYPE

boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable applies to the path-based analysis performed during the **get_timing_paths** and **report_timing** commands when the **-pba_mode** option is specified. This option controls whether or not regular-PBA (path-specific slew propagation) effects are performed during a path-based analysis in addition to AOCVM-PBA.

When the variable is set to *false* (the default), then PrimeTime will perform both regular-PBA and AOCVM-PBA during a path-based analysis. This option will remove the maximum amount of pessimism from the design, but the runtime can be large.

When the variable is set to *true*, then PrimeTime will only perform AOCVM-PBA during a path-based analysis. This option is recommended for fastest runtime in an AOCVM flow.

Note that AOCVM-PBA is applied only if user-specified AOCVM information exists.

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

```
      pt_shell> printvar pba_aocvm_only_mode
```

SEE ALSO

get_timing_paths (2),
read_aocvm (2),
report_timing (2).

pba_derate_list

Specifies path-based derate factors.

TYPE

string

DEFAULT

""

GROUP

timing_variables

DESCRIPTION

This variable specifies a list of **set_timing_derate** commands. Each **set_timing_derate** command specifies a derating factor applicable only during path-based analysis and does not impact regular graph-based timing analysis during **update_timing**. During path-based analysis, path-based derate factors have higher precedence than graph-based derate factors.

This variable is only effective when the **-pba_mode** option on **get_timing_paths** or **report_timing** is set to *path*.

THe option cannot be specified in an AOCVM flow.

In the following example the **pba_derate_list** variable is used to define a late path-based derate factor for cell 'buf0'. It is important to reset the **pba_derate_list** variable after the path-based analysis, so that subsequent path reports are not affected.

```
      pt_shell> set pba_derate_list "set_timing_derate -late 1.12 buf0"
      pt_shell> report_timing -pba_mode path $PATH
      pt_shell> set pba_derate_list ""
```

The next example for **get_timing_paths** is quite similar.

```
      pt_shell> set pba_derate_list "set_timing_derate -late 1.12 buf0"
      pt_shell> get_timing_paths -pba_mode path -to FF4/D
      pt_shell> set pba_derate_list ""
```

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

```
      pt_shell> printvar pba_derate_list
```

SEE ALSO

get_timing_paths (2),

report_timing (2),
set_timing_derate (2).

pba_disable_path_recalculation_limit

Controls whether regular path-based analysis is limited to 10000 paths (the default) or unlimited.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When *false* (the default), PrimeTime will allows only maximum of 10000 paths to be manually recalculated. If more than 10000 paths are provided, only the first 10000 paths will be present in the recalculated paths returned.

If more than 10000 paths must be recalculated, two methods can be used:

- Perform path-based analysis on smaller path sets

- Set this variable to *true*

Removing the limit may be desirable when user understands the runtime implications of unbounded recalculation, but still needs to perform path-based analysis on large collections of paths.

Note that limit does not apply to exhaustive PBA (**-pba_mode** *exhaustive*), as it will always recalculate the minimum but exhaustive set of paths needed to ensure a correct result.

To determine the current value of this variable, type **printvar pba_disable_path_recalculation_limit**.

SEE ALSO

get_timing_paths(2), report_timing(2).

pba_enable_ccs_waveform_propagation

Enables or disables CCS based waveform propagation feature for path-based delay analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When true, enables the CCS based waveform propagation engine to be used in path-based delay analysis. By default, this variable is set to false. In order to use the CCS based waveform propagation engine, in addition to setting this variable to true, you must also make sure that your library contains characterized CCS data.

For uncoupled designs, PrimeTime-SI license is still required to receive the path-based waveform propagation feature. But it is not necessary to set **si_enable_analysis** to true.

For complete information about the CCS based waveform propagation feature, see the PrimeTime-SI User Guide.

To determine the current value of this variable, type **printvar pba_enable_ccs_waveform_propagation**.

SEE ALSO

update_timing (2), **report_timing** (2), **si_enable_analysis** (3).

pba_enable_path_based_physical_exclusivity

This variable controls whether a path-based or stage-based physical exclusivity crosstalk computation is used during path-based analysis of paths involving physically exclusive clocks.

TYPE

int

DEFAULT

false

DESCRIPTION

When set to **false** (the default), each delay calculation stage is evaluated independently of the other stages in the path. For instance, consider two clocks which are physically exclusive, CLK1 and CLK2. For one stage in the path, an aggressor clocked by CLK1 might result in the worst delta delay. For the next stage, an aggressor clocked by CLK2 might result in the worst delta delay. In a stage-based approach, these deltas are both used for the corresponding stages in the path. This approach is runtime efficient, but can possibly result in some pessimism.

When set to **true**, the path is recalculated multiple times to consider each possible victim/aggressor combination across the physically exclusive clocks. In this case, aggressors from CLK1 and CLK2 could not simultaneously attack different stages across the path. This removes the pessimism of the stage-based approach, but at the cost of additional runtime. The recommendation is to leave the default value of false for most analyses, but to set it to true for the final signoff run if additional pessimism removal is desired during path-based analysis.

To determine the current value of this variable, type printvar pba_enable_path_based_physical_exclusivity or echo $pba_enable_path_based_physical_exclusivity.

To determine the current value of this variable, type **printvar pba_enable_path_based_physical_exclusivity** or **echo $pba_enable_path_based_physical_exclusivity**.

SEE ALSO

report_timing(2), get_timing_paths(2).

pba_enable_xtalk_delay_ocv_pessimism_reduction

During the path based analysis reduces the pessimism due to clock on-chip-variation(OCV) on xtalk delay analysis.

TYPE

int

DEFAULT

false

DESCRIPTION

When this is set to **true** (the default is false), during the path based crosstalk delay analysis, PrimeTime SI reduces the pessimism due to impact of clock on-chip-variation(OCV) on the victim and aggressor arrivals. This process is computationally intensive, should be used only when the clock path are long and the on chip variation is large. CRPR must be enabled (by setting **timing_remove_clock_reconvergence_pessimism** to **true**) to use this feature.

To determine the current value of this variable, type **printvar pba_enable_xtalk_delay_ocv_pessimism_reduction** or **echo $pba_enable_xtalk_delay_ocv_pessimism_reduction**.

SEE ALSO

report_timing(2), get_timing_paths(2).
timing_remove_clock_reconvergence_pessimism(3).

pba_exhaustive_endpoint_path_limit

Defines a limit for exhaustive path-based analysis.

TYPE

int (Range: 1 to 2000000)

DEFAULT

25000

GROUP

timing_variables

DESCRIPTION

This variable applies to the exhaustive path-based analysis performed during the **get_timing_paths** or **report_timing** commands when the **-pba_mode exhaustive** option is specified. This exhaustive analysis is computationally intensive and is intended to be used only when the design is approaching signoff.

In certain badly-behaved designs the exhaustive analysis may run for a long time. This variable restricts the exhaustive path search so that the number of paths recalculated at any endpoint is limited. This limit can be adjusted by the user, however increasing the value to a larger number will increase the runtime of the analysis.

There are several other measures that will improve the runtime of the analysis.

- Uniquify path through parallel arcs
- Use conservative values for **-nworst** and **-max_paths**
- Set a realistic threshold for the **-slack_lesser_than option**

When an AOCVM analysis is being performed, there are several additional variable settings that will improve the runtime of the analysis.

- Enable CRPR
- Enable graph-based AOCVM
- Enable PBA AOCVM only mode

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

```
      pt_shell> printvar pba_exhaustive_endpoint_path_limit
```

SEE ALSO

get_timing_paths (2),
report_timing (2),
pba_aocvm_only_mode (3),
timing_aocvm_enable_analysis (3),
timing_remove_clock_reconvergence_pessimism (3),
timing_report_use_worst_parallel_cell_arc (3).

pba_recalculate_full_path

Allow regular path-based analysis to recalculate full clock paths and borrowing path segments.

TYPE

int

DEFAULT

true

DESCRIPTION

When *true* (the default), PrimeTime allows all path-based analysis commands (**report_timing** and **get_timing_paths**) to recalculate full clock paths and borrowing paths in addition to the data paths.

When *false*, PrimeTime blocks recalculation of clock paths and borrowing path portions and the original timing is retained. This allows paths obtained with -path full_clock, -path full_clock_expanded and -trace_latch_borrow to be fully reported, while avoiding recalculation on the borrowing and clock portions of the path. The reason this may be desirable is that with certain circuit topologies, a conservative path-based recalculation of the clock or borrowing path may not be guaranteed. This can happen when there are multiple clock or borrowing paths which can apply to the path. Only the worst pre-recalculation clock or borrowing path is included for recalculation. This may not be the worst path after recalculation.

This variable only applies to regular path-based analysis. It does not apply to AOCVM path-based analysis.

To determine the current value of this variable, type **printvar pba_recalculate_full_path**.

SEE ALSO

report_timing (2), **get_timing_paths** (2).

power_analysis_mode

Sets the power analysis mode

TYPE

fIstringfP

DEFAULT

averaged

GROUP

power_variables

DESCRIPTION

Use this variable to explicitly select the analysis mode for power calculation. Today PrimeTime PX provides three different analysis modes: averaged, time_based and leakage_variation. This variable needs to be set before the first power command, otherwise, the default mode will be assumed. For a particular analysis mode, appropriate activity information must be provided. Allowed values are as follows:

* *averaged* (the default): PrimeTime PX will calculate power based on toggle-rate and state-probability. Only averaged power results will be calculated. In this mode, it can take VCD file, SAIF file as activity input files. Commands **set_switching_activity** and **set_case_analysis** can also be used to set the statistical switching activity on top of the default switching activity. For more information, please check out the man page for *update_power*.

* *time_based*: PrimeTime PX will calculate power based on the events from VCD. Not only averaged power results will be calculated, but also peak powers and time_based power waveforms will be calculated. VCD file must be provided in this mode. Both gate-level VCD and RTL VCD can be specified for this mode. For more information, please check out the man page for *update_power*.

* *leakage_variation*: PrimeTime PX will perform leakage variation analysis. For more information, please check out the man page for **power_enable_leakage_variation_analysis**.

Variable **power_analysis_options** can be changed in one run. However, once changed, all the activity and power data will be removed internally. User needs to provide activity information again before **update_power**.

In addition, command **set_power_analysis_options** can be used to specify the options for power analysis.

SEE ALSO

power_enable_analysis(3), update_power (2), **set_power_analysis_options(2), report_power(2), read_vcd(2), read_saif(2), set_switching_activity(2), set_case_analysis(2), power_enable_leakage_variation_analysis(3), printvar(2).**

power_calc_use_ceff_for_internal_power

Specifies whether to use effective C for internal power calculation.

TYPE

fIbooleanfP

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable controls whether to use effective capacitance in the sense of timing as the output net capacitance parameter when looking up internal power tables during the power calculation stage. If the variable is true, use effective capacitance; otherwise use the total net capacitance.

To determine the current value of this variable, type **printvar power_calc_use_ceff_for_internal_power** or **echo $power_calc_use_ceff_for_internal_power**.

power_check_defaults

Defines the default checks for the **check_power** command.

TYPE

list

DEFAULT

out_of_table_range missing_table missing_function

DESCRIPTION

Defines the default checks to be performed when the **check_power** command is executed without any options. The same default checks are also performed if the **check_power** command is used with **-include** or **-exclude** options. The default check list defined by this variable can be overriden by either redefining it before **check_power** is executed or using the **-override_defaults** option of the **check_power** command.

If an undefined check is specified while redefining this variable, a warning will be issued by the next execution of the **check_power** command.

SEE ALSO

check_power (2).

power_clock_network_include_clock_gating_network

Indicates that clock gating networks are included in the clock network.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects the predefined **clock_network** and **register** power groups. When the variable is *true*, discrete logic structure functioning as clock gating is taken as belonging to the clock_network power group. Only the typical clock gating logic is considered as qualified clock gating network to be included in the clock network. The typical clock gating logic starts from the output of a level sensitive latch driven by the specified clock, possibly goes through a couple of buffers, and comes back to the specified clock network through one of the input pins of an AND or OR gate. The input pin must be a PrimeTime clock check enable pin, either inferred or manually set. Therefore, the results can be affected by clock gating check related commands or variables. When the clock gating network is included in the clock network, the latch is regarded as in the **clock_network** power group, but not in the **register** power group.

To determine the current value of this variable, type **printvar power_clock_network_include_clock_gating_network** or **echo $power_clock_network_include_clock_gating_network**.

SEE ALSO

report_power (2), **create_power_group** (2).

power_clock_network_include_register_clock_pin_power

Indicates whether the register clock pin power is included when reporting clock_network power.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable affects the power report for the predefined **clock_network** and **register** power groups. When *true*, the internal power of registers caused by the toggling of register clock pin when output pin does not toggle is included as **clock_network** power and excluded from **register** power. When *false*, the power is included as **register** power and excluded from **clock_network** power.

To determine the current value of this variable, type **printvar power_clock_network_include_register_clock_pin_power** or **echo $power_clock_network_include_register_clock_pin_power**.

SEE ALSO

report_power (2), **create_power_group** (2).

power_default_static_probability

Specifies the default static probability value.

TYPE

Float

DEFAULT

0.5

DESCRIPTION

The power_default_static_probability variable and the power_default_toggle_rate and power_default_toggle_rate_reference_clock variables are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black-box cells. The power_default_toggle_rate is used to specify the default toggle rate value and the power_default_toggle_rate_reference_clock variable is used to specify how the related clock for default toggle rate is determined.

For other unannoated nets, PrimeTime PX propagates the switching activities of the driving cell inputs based on the cell functionality to derive the switching activity required for power calculations. This mechanism cannot be used for primary inputs and black-box outputs. Instead the following values are used for these type of nets:

- User annotated values are used.

- In some cases, unannotated switching activity values may still be accurately derived, for example, if the net drives a buffer cell and the output of this cell is user annotated, then the user annotated values are used as the default values. Also, if the input is a clock then the clock period and waveform are used to derive the switching activity values.

- If the static probability is not annotated then the value of the power_default_static_probability variable is used for the static probability value.

- If the toggle rate is not user annotated, no mater the static probability is set or not, the following is used for the toggle rate value:

dtr * fclk

where fclk is the frequency of the related clock, and the dtr is the value of the power_default_toggle_rate variable.

The related clock is determined by the value of power_default_toggle_rate_reference_clock. Two values (fastest, related) are allowed for the value of power_default_toggle_rate_reference_clock variable. If the value fastest is specified, the related clock would be simply the fastest clock in the design. If the value related is given, the related clock would depend on which clock domain the net belongs to.

The value of power_default_static_probability should be between 0.0 and 1.0, both inclusive. The value of power_default_toggle_rate should be greater or equal to 0.0. Also, if the value of power_default_static_probability is 0.0 or 1.0, then the value of power_default_toggle_rate should be 0.0. If the value of power_default_toggle_rate is 0.0, then the value of power_default_static_probability should be either 0.0 or 1.0.

The default value of both variables is 0.5.

SEE ALSO

power_default_toggle_rate (3), **power_default_toggle_rate_reference_clock** (3), **set_switching_activity** (2),

power_default_toggle_rate

Specifies the default toggle rate value.

TYPE

Float

DEFAULT

0.1

DESCRIPTION

The power_default_toggle variable and the power_default_toggle_rate_reference_clock and power_default_static_probability variables are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black-box cells.

For other unannoated nets, PrimeTime PX will propagate the switching activities of the driving cell inputs based on the cell functionality to derive the switching activity required for power calculations. This mechanism cannot be used for primary inputs and black-box outputs. Instead the following values are used for these type of nets:

- User annotated values are used.

- In some cases, unannotated switching activity values may still be accurately derived, for example, if the net drives a buffer cell and the output of this cell is user annotated, then the user annotated values are used as the default values. Also, if the input is a clock then the clock period and waveform are used to derive the switching activity values.

- If the static probability is not annotated then the value of the power_default_static_probability variable is used for the static probability value.

- If the toggle rate is not user annotated, no mater the static probability is set or not, the following is used for the toggle rate value:

dtr * fclk

where fclk is the frequency of the related clock, and dtr is the value of the power_default_toggle_rate variable.

The related clock is determined by the value of power_default_toggle_rate_reference_clock. Two values (fastest, related) are allowed for the value of power_default_toggle_rate_reference_clock variable. If the value fastest is specified, the related clock would be simply the fastest clock in the design. If the value related is given, the related clock would depend on which clock domain the net belongs to.

The value of power_default_static_probability should be between 0.0 and 1.0, both inclusive. The value of power_default_toggle_rate should be greater or equal to 0.0. Also, if the value of power_default_static_probability is 0.0 or 1.0, then the value of power_default_toggle_rate should be 0.0. If the value of power_default_toggle_rate is 0.0, then the value of power_default_static_probability should be either 0.0 or 1.0.

The default value of power_default_static_probability is 0.5 and the default value of power_default_toggle_rate is 0.1.

SEE ALSO

power_default_toggle_rate_reference_clock (3), **power_default_static_probability** (3), **set_switching_activity** (2),

power_default_toggle_rate_reference_clock

Specifies how the related clock for default toggle rate is determined.

TYPE

string **power_default_toggle_rate_reference_clock**

DEFAULT

related

DESCRIPTION

The power_default_toggle_rate_reference_clock variable and the power_default_toggle_rate and power_default_static_probability variables are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black-box cells. The power_default_toggle_rate is used to specify the default toggle rate value and the power_default_toggle_rate_reference_clock variable is used to specify how the related clock for default toggle rate is determined.

For other unannoated nets, PrimeTime PX propagates the switching activities of the driving cell inputs based on the cell functionality to derive the switching activity required for power calculations. This mechanism cannot be used for primary inputs and black-box outputs. Instead the following values are used for these type of nets:

- User annotated values are used.

- In some cases, unannotated switching activity values may still be accurately derived, for example, if the net drives a buffer cell and the output of this cell is user annotated, then the user annotated values are used as the default values. Also, if the input is a clock then the clock period and waveform are used to derive the switching activity values.

- If the static probability is not annotated then the value of the power_default_static_probability variable is used for the static probability value.

- If the toggle rate is not user annotated, no mater the static probability is set or not, the following is used for the toggle rate value:

dtr * fclk

where fclk is the frequency of the related clock, and the dtr is the value of the power_default_toggle_rate variable.

The related clock is determined by the value of power_default_toggle_rate_reference_clock. Two values (fastest, related) are allowed for the value of power_default_toggle_rate_reference_clock variable. If the value fastest is specified, the related clock would be simply the fastest clock in the design. If the value related is given, the related clock would depend on which clock domain the net belongs to.

The value of power_default_static_probability should be between 0.0 and 1.0, both inclusive. The value of power_default_toggle_rate should be greater or equal to 0.0. Also, if the value of power_default_static_probability is 0.0 or 1.0, then the value of power_default_toggle_rate should be 0.0. If the value of power_default_toggle_rate is 0.0, then the value of power_default_static_probability should be either 0.0 or 1.0.

The default value of both variables is 0.5.

SEE ALSO

power_default_static_probability (3), **power_default_toggle_rate** (3), **set_switching_activity** (2).

power_domains_compatibility

Indicates whether to revert to power domains mode and disable UPF mode.

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

Indicates whether to revert to power domains mode and disable UPF mode. Power domains are the previous (PrimeTime version Z-2007.06) method of specifying virtual power network and power intent. Starting with version A-2007.12, PrimeTime will be in UPF mode by default and all power domain commands will be unavailable.

When you set this variable to TRUE, - All UPF commands are disabled - Power domain commands are enabled - All designs and their annotations are removed from memory.

This variable is equivalent to the Design Compiler shell startup option -upf_mode.

To determine the current value of this variable, type **printvar power_domains_compatibility** or **echo $power_domains_compatibility**.

SEE ALSO

To list commands available in UPF mode, use **help upf** To list commands available in power domains mode, use **help "power domains"**

power_enable_analysis

Enables or disables PrimeTime PX, which provides power analysis.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When true, enables PrimeTime PX, so that user can perform power analysis. Without this varibale set to TRUE user will not be able to see power related data. By default, PrimeTime PX is disabled; this variable is set to false.

If you set this variable to true and enable PrimeTime PX, you must also do the following:

1. Obtain a PrimeTime PX license. You cannot use PrimeTime PX without a license.

For the current value of this variable, type **printvar power_enable_analysis**.

SEE ALSO

printvar (2),

power_enable_leakage_variation_analysis

Enables or disables the leakage variation feature in PrimeTime PX.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When true, enables the leakage variation feature in PrimeTime PX. Without this varibale set to TRUE user will not be able to compute or report leakage variation information.

In order to successfully use the leakage variation feature, the following other variables must also be set to TRUE:

power_enable_analysis variation_enable_analysis

In order to successfully use the leakage variation feature, libraries containing leakage variation information must be used. see command **set_variation_library**.

Also, variation commands must be issued to describe the variation of the parameters. See commands **create_distribution**, **set_variation**, **set_variation_correlation** for more information on how to do this.

In order to use the leakage variation feature, the user must have a **PrimeTime-PX** license, as well as a **PrimeTime-VX** license. The **PrimeTime-New-Technology** license must also be available in order to use the leakage variation feature.

For the current value of this variable, type **printvar power_enable_leakage_variation_analysis**.

SEE ALSO

printvar (2), **set_variation_library** (2), **set_variation** (2), **power_enable_analysis** (3), **variation_enable_analysis** (3),

power_estimate_power_for_unmatched_event

Controls to estimate power when no table is found in the library to match a certain event.

TYPE

fIbooleanfP

DEFAULT

true

GROUP

power_variables

DESCRIPTION

Sometimes, when an output pin toggles, PrimeTime PX cannot find a matching table in the library based on the current state of the cell. This variable controls whether PrimeTime PX should skip this event without power contribution or try to estimate a power number for it according to all the tables in the library relating to this output pin.

To determine the current value of this variable, type **printvar power_estimate_power_for_unmatched_event** or **echo $power_estimate_power_for_unmatched_event**.

power_include_initial_x_transitions

Controls to count x power in the power up initialization stage.

TYPE

fIbooleanfP

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Initially, if user does not set a logic value to a certain net, PrimeTime PX assume it is X. In the later stage, the value becomes 0 or 1. This variable controls whether PrimeTime PX should count the power caused by the X->0 or X->1 toggle.

To determine the current value of this variable, type **printvar power_include_initial_x_transitions** or **echo $power_include_initial_x_transitions**.

power_limit_extrapolation_range

Specifies if extrapolation will be limited to certain range for internal power calculation.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable specifies if extrapolation will be limited to certain range for internal power calculation. By default, PrimeTime PX does extrapolation with no limit if the data point for internal power table lookup is out-of-range. This applies for both nlpm (non-linear power model) and sppm (scalable polynomial power model) types of internal power table. If this variable is set to true, PrimeTime PX stops extrapolation as follows:

for nlpm power table: PrimeTime PX stops extrapolation at one additional index grid.
for sppm power table: PrimeTime PX stops extrapolation at the specified range.

For the current value of this variable, type **printvar power_limit_extrapolation_range**.

SEE ALSO

printvar (2),

power_match_state_for_logic_x

Specifies logic x interpretation.

TYPE

fIcharfP

DEFAULT

0

GROUP

power_variables

DESCRIPTION

This variable specifies how PrimeTime PX interprets logic x in the boolean function of "when" state of a power table. The command uses the following settings:

0 -- regards x as zero (0)

1 -- regards x as one (1)

x/X -- regards x as neither 0 nor 1, i.e. whenever there is any x logic in it, the boolean function will be evaluated false.

SEE ALSO

printvar (2),

power_model_preference

Specifies the power model preference if the library contains both NLPM and CCS power data.

TYPE

fIstringfP

DEFAULT

ccs

GROUP

power_variables

DESCRIPTION

A library can contain either CCS Power, NLPM or both type of data within a cell definition. Use this variable to specify the power model preference if the library contains both NLPM and CCS power data. Allowed values are as follows:

* *ccs* (the default): PrimeTime PX will use CCS Power data in library (if present) to calculate both static and dynamic power. If CCS Power data is not found, PrimeTime PX will use NLPM data.

* *nlpm*: If this variable is set to "nlpm", PrimeTime PX will use NLPM data as preference. If NLPM data is not found, PrimeTime PX will use CCS Power data.

If neither CCS Power and NLPM data is found for a cell in the library, this cell is not characterized for power analysis.

SEE ALSO

printvar (2).

power_rail_output_file

Specifies the output file name for dumping out power information for PrimeRail users.

TYPE

string

DEFAULT

"" (empty)

GROUP

power_variables

DESCRIPTION

This variable is provided for PrimeRail users. If the file name is provided with this variable, during power calculation relevant power information will be dumped into the file provided. Please note that information is dumped in a binary format. This file is then read by PrimeRail.

Starting from 2008.12 release, **update_power** can perform both average and peak power analysis. Variable **power_analysis_mode** can be used to specify the power analysis mode to perform. PrimeTime PX can perform different types of power analysis based on the mode setting. The default power analysis mode is "averaged". For more information, please check out the man page for **power_analysis_mode**.

Command **set_power_analysis_options** can be used to specify the options for power analysis. It must be set before **update_power** to take effects in power analysis. Issuing command **set_power_analysis_options** with no option can reset all the power analysis options to default. Use command **report_power_analysis_options** to get the current analysis option settings. please check out the man page for **set_power_analysis_options** for more information.

PrimeTime PX can consume either time-based (e.g. VCD file) or statistical switching activity information (e.g. SAIF file) for power calculation. For a particular analysis mode, appropriate activity information must be provided. For example, in time_based power analysis mode, a VCD file must be provided. In averaged power analysis mode, any form of switching activity information is accepted. You can specify a VCD file by using command **read_vcd**. The statistical switching activity can be specified by using command **read_saif** or **set_switching_activity**.

For the current value of this variable, type **printvar power_rail_output_file**.

SEE ALSO

printvar (2), **update_power** (2), **power_analysis_mode** (3), **set_power_analysis_options**

(2), **report_power_analysis_options** (2).

power_read_activity_ignore_case

Use the *power_read_activity_ignore_case* variable instead of the *power_read_vcd_ignore_case* variable to control ignore case when reading activity files.

TYPE

fIbooleanfP

DEFAULT

true

GROUP

power_variables

DESCRIPTION

The fipower_read_vcd_ignore_case variable has been replaced with the *power_read_activity_ignore_case* variable. For backward compatibility, the fipower_read_vcd_ignore_case variable is an alias for the *power_read_activity_ignore_case* variable.

The *power_read_activity_ignore_case* variable controls match pin, net and cell names from VCD or SAIF file and those from the design case sensitively if the value of this variable is *false* or design case insensitively if the value is *true*. This variable also affects the **set_rtl_to_gate_name** command.

To determine the current value of the active variable, type **printvar power_read_activity_ignore_case** or **echo $power_read_activity_ignore_case**.

SEE ALSO

printvar(2) set_rtl_to_gate_name(2) read_saif(2) read_vcd(2)

power_read_activity_ignore_case

Use the *power_read_activity_ignore_case* variable instead of the *power_read_vcd_ignore_case* variable to control ignore case when reading activity files.

TYPE

fIbooleanfP

DEFAULT

true

GROUP

power_variables

DESCRIPTION

The fipower_read_vcd_ignore_case variable has been replaced with the *power_read_activity_ignore_case* variable. For backward compatibility, the fipower_read_vcd_ignore_case variable is an alias for the *power_read_activity_ignore_case* variable.

The *power_read_activity_ignore_case* variable controls match pin, net and cell names from VCD or SAIF file and those from the design case sensitively if the value of this variable is *false* or design case insensitively if the value is *true*. This variable also affects the **set_rtl_to_gate_name** command.

To determine the current value of the active variable, type **printvar power_read_activity_ignore_case** or **echo $power_read_activity_ignore_case**.

SEE ALSO

printvar(2) set_rtl_to_gate_name(2) read_saif(2) read_vcd(2)

power_report_leakage_breakdowns

Controls to report leakage components.

TYPE

fIbooleanfP

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable is to control whether report_power will print out leakage power components or not. By default, the variable value is false, which means report_power will not report leakage power components, i.e., only total leakage is reported. If the variable is set to true, then intrinsic leakage and gate leakage will also be reported in addition to the total leakge in the summary report and the cell based power report.

SEE ALSO

printvar (2), **report_power** (2).

power_reset_negative_extrapolation_value

Resets negative extrapolated energy value from library to zero.

TYPE

fIbooleanfP

DEFAULT

false

GROUP

pwr_variables

DESCRIPTION

The variable **power_reset_negative_extrapolation_value** will be used to reset negative extrapolated energy value from library to zero, if the energy value for the boundary points is positive. If this variable is set to true then the negative extrapolated energy value will be reset to zero.

In some cases the values of variables (mostly output capacitance and input slew), which is used for extracting the energy number from library energy tables is out of range, i.e., the values may be greater or smaller than the boundary points. In this scenario extrapolation techniques are used for deriving the energy number from library energy tables. If the variable values are too small or too big then the derived energy number may come out to be negative. However, the energy number corresponding to the boundary points may be positive. Using the negative energy number given the fact that the energy number corresponding to boundary points is positive will result in incorrect energy numbers. To resolve this problem, use variable **power_reset_negative_extrapolation_value**, which if set to true, will reset the negative extrapolated energy numbers to zero, if the energy number for boundary points is positive.

To determine the current value of this variable, type **printvar power_reset_negative_extrapolation_value** or **echo $power_reset_negative_extrapolation_value**.

power_reset_negative_internal_power

Resets the negative internal power to zero.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Resets the negative internal power to zero. This variable can be used if the user is confident about the accuracy of the power tables. Furthermore, if the values of capacitance and input slew values are out of range from range specified in the power tables, then due to extra/intrapolation the internal energy number can be negative. This variable gives the user the choice to reset these negative internal energy numbers to zero.

For the current value of this variable, type **printvar power_reset_negative_internal_power**.

SEE ALSO

printvar (2),

power_scale_dynamic_power_at_power_off

Indicates if the dynamic power will be scaled according to the static probability of the corresponding power supply net. As default, this variable is set to false. Only leakage power is scaled by the power-on probability.

TYPE

fIbooleanfP

DEFAULT

false

GROUP

power_variables

DESCRIPTION

The statistical activity information can be input from SAIF file, set_switching_activity/set_case_analysis commands, or from default settings and propagated through the whole design. PrimeTime PX is built with power management awareness and can reflect power saving technology used in the design in the calculated power results. If the power supply net is switched off during simulation, PrimeTime PX can report the correct dynamic and static (leakage) power based on the statistical activity information.

If the given statistical activity information does not contain any toggle happened at power-off state, only leakage power will be scaled by the power-on probability. This is the default behavior. However, if the input activity information includes toggles happened at power-off state, then both dynamic and leakage power will be scaled. Under such situation, user needs to set <b>power_scale_dynamic_power_at_power_off</b> to true, so that PrimeTime PX will apply the scaling to dynamic power as well.

This variable has no effect if there is no power switch boolean function defined. Also it only applies to SAIF and vector-free flows. It has no effect for VCD flow. As in VCD flow, PrimeTime PX monitors the status of the power supply net and determines the power consumption at event basis.

To determine the current value of this variable, type **printvar power_scale_dynamic_power_at_power_off** or **echo $power_scale_dynamic_power_at_power_off**.

SEE ALSO

read_saif (2), **set_switching_activity** (2), **set_case_analysis** (2).

power_table_include_switching_power

Indicate whether power tables in technology library include switch power.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable is used to indicate whether the NLPM power tables in technology library included switch power components. There can be 2 types of NLPM internal power tables from characterization. One type contains pure internal energy. Another type contains (total energy - 0.5 * switching energy). PrimeTime PX supports the second format by default. For average power analysis, the difference of these two types of characterization does not have significant impact on the results. The effects of added switching energy canceled out by events of opposite edges. For power waveform generation, the added 0.5 switching energy will make the difference, especially for single cell or very small designs. For correlation and characterization verification purpose, small designs or single cell designs may be used and the differences can be significant.

When the variable is set to "false", it means the power tables in the technology library are of the first type, i.e., the values in the tables are pure internal energy. When it is set to "true", the values in power tables are of the second type. PrimeTime PX will choose the proper power calculation approach based on this variable.

When using CCS Power tables, this variable should alway be set to "false".

power_x_transition_derate_factor

Set the scale factor for X-transition power.

TYPE

fIfloatfP

DEFAULT

0.5

GROUP

power_variables

DESCRIPTION

This variable controls a scale factor for X-transition power.

SEE ALSO

printvar (2),

pt_ilm_dir

Specifies a directory for PrimeTime to create ILM related files.

TYPE

fIstringfP

DEFAULT

(current directory)

DESCRIPTION

Specifies a directory for PrimeTime to create ILM related files. By default, the value is ".", the current directory. You can set this variable to any directory, such as /u/primetime/ilm.

To determine the current value of this variable, type **printvar pt_ilm_dir** or **echo $pt_ilm_dir**

SEE ALSO

create_ilm(2)
create_si_context(2)
write_arrival_annotations(2)

pt_shell_mode

Describes the mode of operation of the current shell.

TYPE

string

DESCRIPTION

This read-only variable describes the mode of operation of the current shell. The mode ’primetime’ indicates that the current PrimeTime shell was launched in non multi_scenario mode. The mode ’primetime_master’indicates that the current shell was launched by the user with the -multi_scenario option. The mode ’primetime_slave’ indicates that the current shell was launched by the create_distributed_farm command. The **pt_shell_mode** variable is usefull in scripts or setup files which are to be sourced by both the master and the slave.

To determine the current value of this variable, type **printvar pt_shell_mode**

pt_tmp_dir

Directory that PrimeTime uses for temporary storage.

TYPE

fIstringfP

DEFAULT

/tmp (the local /tmp partition)

DESCRIPTION

Specifies a directory for PrimeTime to use as temporary storage. By default, the value is "/tmp". You can set this variable to any directory with proper read/write permissions, such as ".", the current directory.

To determine the current value of this variable, type **printvar pt_tmp_dir** or **echo $pt_tmp_dir**

SEE ALSO

set_program_options (2)

ptxr_root

Specifies an alternative installation root path for PrimeTime to look for the executables required by PrimeTime External Reader (ptxr).

TYPE

fIstringfP

DEFAULT

By default, this variable is the same as the root path where PrimeTime program is installed.

DESCRIPTION

When set to a different path from the default PrimeTime installation root, this variable contains a path name to the executables specific to PrimeTime external reader (ptxr). Instead of using the reader programs installed within PrimeTime's root path, this provides user with the flexibility of supplying an alternative program that is equivalent to the natively installed executable to achieve reading of file formats that are only supported by ptxr.

Because this alternative root path is outside of PrimeTime, the availability and completeness of that installation is not garanteed by PrimeTime. When the expected ptxr executables cannot be located within the user specified root path, PrimeTime will fall back and proceed with the natively installed program.

This variable is intended for use only when the natively installed ptxr programs do not work with certain files of supported formats. Most often it happens when trying to read files generated by a newer version of synopsys tool such as DesignCompiler or PhysicalCompiler.

The following example shows how to use the ptxr_root to specify an alternative ptxr program.

```
pt_shell> set ptxr_root /tools/DC2004.12-SP1/snps/synopsys
/tools/DC2004.12-SP1/snps/synopsys
pt_shell> read_ddc new_design.ddc
Information: Using user set ptxr_root '/tools/DC2004.12-SP1/snps/synopsys'.
Beginning read_ddc...
```

It also should be noted that when this variable is set, all down stream file reading which requires ptxr will use the reader from the specified path unless it is explicitly set back to the defalut.

SEE ALSO

read_lib (2), **read_vhdl** (2). **read_ddc** (2). **ptxr_setup_file** (3).

ptxr_setup_file

Specifies a setup file to be read by the PrimeTime External Reader (ptxr) instead of home and local .synopsys_dc.setup files.

TYPE

fIstringfP

DEFAULT

By default, this variable does not exist.

DESCRIPTION

When set, this variable contains a pathname to a setup file specific to the PrimeTime external reader (ptxr). By default, this variable does not exist until you set it with a value.

This variable is intended for use only if you are reading a netlist using ptxr, by executing one of the following:

- **read_verilog -hdl_compiler**
- **read_vhdl**
- **read_ddc**

The ptxr program uses the same reader as that used by Design Compiler. When you execute one of the above commands, the ptxr program by default reads your .synopsys_dc.setup files (not .synopsys_pt.setup), including the system, home, and local setup files, to access needed variables. You cannot disable reading of system .synopsys.dc.setup files, but you can substitute a ptxr-specific setup file for the home and local setup files by setting the **ptxr_setup_file** variable with the pathname of a ptxr-specific setup file you create.

You write the **ptxr_setup_file** in Tcl. The file can contain a very limited set of commands: comments, blank lines, and variable assignments, as in the following example for a file named my_ptxr.setup:

```
         # My ptxr_setup_file
set bus_naming_style "%s(%d)"
set bus_extraction_style "%s[%d:%d]"
```

The following example shows how to use the ptxr setup file when reading a Verilog netlist with ptxr. First, you set the **ptxr_setup_file** variable with the filename my_ptxr.setup. Next, you invoke the **read_verilog** command using the **-hdl_compiler** option.

```
pt_shell> set ptxr_setup_file my_ptxr.setup
my_ptxr.setup
pt_shell> read_verilog -hdl_compiler module1.v
```

To discontinue using the ptxr setup file, unset the **ptxr_setup_file** variable, as follows:

```
pt_shell> unset ptxr_setup_file
```

SEE ALSO

read_verilog (2), **read_vhdl** (2). **read_ddc** (2).

rc_adjust_rd_when_less_than_rnet

Enables or disables overriding of library-derived drive resistance when it is much less than the dynamic RC network impedance to ground.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true (the default), PrimeTime checks the library-derived drive resistance, and if it is less than the dynamic RC network impedance to ground by an amount equal to or greater than the threshold value contained in the variable **rc_rd_less_than_rnet_threshold** (default 0.45), PrimeTime adjusts the drive resistance using an empirical formula. To disable the checking and adjustment, set the **rc_adjust_rd_when_less_than_rnet** variable to false.

When the library-derived drive resistance is much less than the dynamic RC network impedance to ground, the behavior of the resistor-based driver model can deviate from that of transistors. In this case, PrimeTime replaces the drive resistance with a value obtained by using an empirical formula to improve accuracy, and issues the RC-009 message. This entire process of checking, detection, replacement, and issuing of the message is referred to as the "RC-009 condition".

This variable is one of a set of four variables relevant to the RC-009 condition. The other three are effective only when **rc_adjust_rd_when_less_than_rnet** is true, and are as follows:

• **rc_degrade_min_slew_when_rd_less_than_rnet** determines whether or not slew degradation is used in min analysis mode when the RC-009 condition occurs. The default is false. To use slew degradation during the RC-009 condition, set this variable to true.

• **rc_filter_rd_less_than_rnet** determines whether the RC-009 message is issued only when a network delay is greater than the corresponding driver transition time. The default is true. To receive RC-009 messages every time PrimeTime overrides the drive resistance, set this variable to false.

• **rc_rd_less_than_rnet_threshold** specifies the threshold beyond which PrimeTime overrides the library-derived drive resistance with an empirical formula. The default is 0.45. If there is a reason why the default is not appropriate in your situation, you can set this variable to another value.

For more information, see the manual page of the RC-009 warning message.

To determine the current value of this variable, type **printvar rc_adjust_rd_when_less_than_rnet** or **echo $rc_adjust_rd_when_less_than_rnet**.

SEE ALSO

rc_degrade_min_slew_when_rd_less_than_rnet (3), **rc_filter_rd_less_than_rnet** (3), **rc_rd_less_than_rnet_threshold** (3); **RC-009** (n).

rc_always_use_max_pin_cap

Specifies whether to override the use of pin-capacitances from the min library during min RC delay-calculation and use the pin-capacitances from the max library instead. This functionality is provided only to obtain backward compatibility with older releases of PrimeTime.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

As of the 2002.09 release of PrimeTime, support for min pin-capacitance during min RC delay-calculation is provided. To achieve backward compatibility with previous releases of PrimeTime, set the **rc_always_use_max_pin_cap** variable to true.

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

```
pt_shell> printvar rc_always_use_max_pin_cap
```

SEE ALSO

set_min_library (2).

rc_cache_min_max_rise_fall_ceff

Specifies whether to cache min/max rise/fall values of effective capacitance computed during RC delay calculation.

TYPE

boolean

DEFAULT

false

DESCRIPTION

Specifies whether to cache min/max rise/fall values of effective capacitance computed during RC delay calculation. These cached values may be queried via attributes on driver pins and ports with driving cells.

When the **rc_cache_min_max_rise_fall_ceff** shell variable is set true before a timing update occurs, the effective-capacitance (also known as C-effective) associated with propagated driver behavior is cached according to min/max and rise/fall characteristics.

There are four driver-pin/port attributes that can be queried to obtain the cached values: **cached_ceff_max_rise**, **cached_ceff_min_rise**, **cached_ceff_max_fall**, and **cached_ceff_min_fall**. These cached attributes are useful for obtaining the worst-case C-effectives for every driver in the design. Note that the other C-effective attributes (that is, **effective_capacitance_min**, **effective_capacitance_max**, **ceff_params_min**, and **ceff_params_max**) are computed at query time and, thus, take considerably more runtime.

The values of the cached attributes depend on the selected slew-propagation mode. See the **timing_slew_propagation_mode** shell variable man page for more information about slew propagation.

Note that if the **rc_cache_min_max_rise_fall_ceff** shell variable is false during the most-recent timing update, the cached values are not created or updated. This means that the attribute values might be nonexistent or invalid.

The following **tcl** code show how to use the attribute query results only when the attributes exist.

```
set ceff \
[get_attribute -quiet $obj ceff_min_rise]
if {[string length $ceff] != 0} {
  ...
}
```

The cached values are removed only when you execute the **remove_annotated_parasitics** command or when a netlist-editing command has similar reason to remove annotated parasitics.

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

```
pt_shell> printvar \
rc_cache_min_max_rise_fall_ceff
```

SEE ALSO

remove_annotated_parasitics (2), **timing_slew_propagation_mode** (3).

rc_ceff_delay_min_diff_ps

Specifies a tolerance for determining when the effective-capacitance calculation has converged during RC delay calculation. Please read the description below for important issues regarding the use of this variable.

TYPE

double

DEFAULT

0.25

DESCRIPTION

When there is a problem with library timing data, numerous RC-004 warning messages can occur with the stated reason, "because the library data is inconsistent with a linear-driver model." Usually these failures are caused by incorrect threshold settings or artificial library data (for example, merged tables), but occasionally the library data will have been generated with insufficiently accurate simulation settings.

In this last case, interpolation error within inaccurate library tables can lead to convergence problems during the effective-capacitance calculation. The shell variable **rc_ceff_delay_min_diff_ps** is available to mitigate these problems as a temporary measure, until the library data can be fixed.

Small increases in the value of this variable are sufficient to discern its impact. If the majority of the RC-004 messages do not go away after doubling or quadrupling the default value, the cause of the messages is much more likely to be due to incorrect threshold settings or artificial library data.

PLEASE NOTE: Increasing the value of this variable can help avoid falling back to lumped RC delay calculation in some RC-004 cases, but it can also adversely impact the accuracy for those RC delay calculations without RC-004 messages. Use of this variable should be viewed as a temporary measure until the library data can be fixed.

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

```
pt_shell> printvar rc_ceff_delay_min_diff_ps
```

SEE ALSO

RC-004 (n).

rc_ceff_use_delay_reference_at_cpin

Specifies whether to compute C-effective using a driver delay relative to that for the output pin capacitance. This can improve accuracy for drivers with many internal stages (e.g. extracted timing models), but it requires library data characterized all the way to down to the output pin capacitance.

TYPE

boolean

DEFAULT

false

DESCRIPTION

PrimeTime adjusts C-effective to match library and RC-network delays to within a small percentage. When the library delay is very large, as would be the case for a driver with many internal stages (e.g. an extracted timing model), this criterion can be met without a sufficient number of C-effective iterations.

If **rc_ceff_use_delay_reference_at_cpin** is set true, then PrimeTime will exclude the portion of the library delay due to output pin capacitance from the C-effective convergence criterion. This will allow a sufficient number of iterations to occur.

Note that if **rc_ceff_use_delay_reference_at_cpin** is true, then the library data must be characterized all the way down to the output pin capacitance.

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

```
pt_shell> printvar rc_ceff_use_delay_reference_at_cpin
```

SEE ALSO

rc_ceff_delay_min_diff_ps (3).

rc_create_and_cache_pi_models

Specifies whether to create and cache pi-model reductions of annotated detailed parasitics during timing updates. The resulting pi-models can be written out to a file using the **write_pi_parasitics** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Pi-models are highly reduced forms of annotated detailed parasitics. PrimeTime will create and cache pi-models if the shell variable **rc_create_and_cache_pi_models** is set *true* before a timing update occurs. If this variable is *false* when **write_pi_parasitics** is issued, the variable is set to *true*, and a full timing update is launched.

The cached pi-models are only removed when either the **remove_annotated_parasitics** command is issued or when a netlist editing command has similar cause to remove annotated parasitics.

This variable will be obsoleted from future releases of PrimeTime starting with 2009.12 on all platforms.

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

```
pt_shell> printvar rc_create_and_cache_pi_models
```

SEE ALSO

write_pi_parasitics (2).

rc_degrade_min_slew_when_rd_less_than_rnet

Enables or disables the use of slew degradation in min analysis mode during the RC-009 condition.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When false (the default), PrimeTime does not use slew degradation through RC networks in min analysis mode during the RC-009 condition. When true, PrimeTime uses slew degradation during the RC-009 condition. This variable is effective only if the **rc_adjust_rd_when_less_than_rnet** variable is true.

The "RC-009 condition" means a condition in which PrimeTime checks the library-derived drive resistance, and if it is less than the dynamic RC network impedance to ground by an amount equal to or greater than the value of the **rc_rd_less_than_rnet_threshold** variable, PrimeTime adjusts the drive resistance using an empirical formula to improve accuracy, and issues the RC-009 message. In case this improved accuracy is not sufficient, PrimeTime provides extra pessimism by not using slew degradation in min analysis mode; however, superfluous min delay violations could occur as a side effect. You can keep slew degradation on in min analysis mode after you have qualified the RC-009 methodology for your accuracy requirements, by setting this variable to true.

rc_degrade_min_slew_when_rd_less_than_rnet is one of a set of four variables relevant to the RC-009 condition. The other three are as follows:

- **rc_adjust_rd_when_less_than_rnet** enables or disables the RC-009 condition; the default is true. When this variable is set to false, PrimeTime does not check the drive resistance, and the values of the other related variables do not matter.

- **rc_filter_rd_less_than_rnet** determines whether the RC-009 message is issued only when a network delay is greater than the corresponding driver transition time. The default is true. To receive RC-009 messages every time PrimeTime overrides the drive resistance, set this variable to false. This variable has no effect if **rc_adjust_rd_when_less_than_rnet** is false.

- **rc_rd_less_than_rnet_threshold** specifies the threshold beyond which PrimeTime overrides the library-derived drive resistance with an empirical formula. The default is 0.45 ohms. You can override this default by setting the variable to another value. This variable has no effect if **rc_adjust_rd_when_less_than_rnet** is false.

Note: If **rc_degrade_slew_when_rd_less_than_rnet** is false while **rc_filter_rd_less_than_rnet** is true, the RC-009 message is not issued.

For more information, see the manual page of the RC-009 warning message.

To determine the current value of this variable, type **printvar rc_degrade_min_slew_when_rd_less_than_rnet** or **echo $rc_degrade_min_slew_when_rd_less_than_rnet**.

SEE ALSO

rc_ajust_rd_when_less_than_rnet (3), **rc_filter_rd_less_than_rnet** (3), **rc_rd_less_than_rnet_threshold** (3); **RC-009** (n).

rc_driver_count_threshold_for_fast_multidrive_analysis

Specifies how many network drivers beyond which will necessitate fast multi-drive analysis.

TYPE

integer

DEFAULT

9

DESCRIPTION

The runtime of full-accuracy RC delay-calculation scales as the product #drivers x #loads. PrimeTime has a special ’fast multi-drive analysis’ feature that can reduce runtime by many orders of magnitude for massively multi-driven networks. However, the analysis assumes that all driver timing arcs, operating-conditions, input-slews, and input-skews are identical. The analysis also assumes the effective-capacitance is the same for all drivers.

This analysis mode is activated when the number of strong (i.e. non-3state) drivers on a net exceeds the number specified with the shell variable **rc_driver_count_threshold_for_fast_multidrive_analysis**. Setting this variable to zero shuts the feature off completely.

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

```
pt_shell> printvar rc_driver_count_threshold_for_fast_multidrive_analysis
```

SEE ALSO

RC-010 (n).

rc_driver_model_max_error_pct

Specifies the maximum error tolerated in a driver model used in RC effective-capacitance calculations.

TYPE

double

DEFAULT

16.0

DESCRIPTION

When a driving cell is connected to a network with annotated parasitics, an effective capacitance is computed for use with lumped-load based library data. For a candidate value of effective capacitance, PrimeTime builds a driver model that matches the library's delay, slew, and sensitivity to output capacitance. Since the PrimeTime driver model is an abstract approximation of the actual transistor behavior, it is usually impossible to match these criteria exactly. Some error tolerance must be used.

The total allowable error tolerance over all criteria can be specified with **rc_driver_model_max_error_pct**. PrimeTime tries initially to build a driver model with 1% error and gradually relaxes that goal to the value of **rc_driver_model_max_error_pct**.

The value of **rc_driver_model_max_error_pct** is ignored if it is set to a value less than 1%. The value is specified as a percentage, so a desired error tolerance of 10% should be specified as 10.0.

You can use the **report_driver_model** command to examine the driver model error, library data, and matching criteria for a given operating point.

If the RC effective-capacitance calculation fails because the driver model error is greater than **rc_driver_model_max_error_pct**, the warning message RC-004 (single drive) or RC-007 (multidrive) will be issued.

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

pt_shell> **printvar rc_driver_model_max_error_pct**

SEE ALSO

report_driver_model (2), **RC-004** (n), **RC-007** (n).

rc_driver_model_mode

Specifies which driver model type to use for RC delay calculation.

TYPE

string

DEFAULT

advanced

DESCRIPTION

PrimeTime supports two types of driver models for RC delay calculation, basic and advanced. The basic model is derived from the conventional delay and slew library schema, while the advanced model is derived from a new schema. The advanced model has many advantages, one of which is the solution to the problem described by the RC-009 warning message. The advanced driver model is part of the Synopsys Composite Current-Source (CCS) model.

When the shell variable **rc_driver_model_mode** is set to *basic*, RC delay calculation will always use driver models derived from the conventional delay and slew schema present in design libraries. When set to *advanced*, RC delay calculation will use the advanced driver model if data for it is present. The **report_delay_calculation** command used on a cell arc will show the message "Advanced driver-modeling used" as appropriate.

When the shell variable **rc_driver_model_mode** is set to *basic*, and the variable **rc_receiver_model_mode** is set to *advanced*, PrimeTime will use the advanced voltage-dependent capacitance models to derive an equivalent single capacitance dependent only on the rise, fall, min or max arc condition and these equivalent capacitances will be used in analysis instead of the pin capacitances from the library. Please check the manpage for variable **rc_receiver_model_mode** for additional details.

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

```
pt_shell> printvar rc_driver_model_mode
```

SEE ALSO

rc_receiver_model_mode (3), **report_delay_calculation** (2).

rc_filter_rd_less_than_rnet

Enables or disables suppressing the display of RC-009 messages when the network delay is less than the corresponding driver transition time.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true (the default), PrimeTime displays the RC-009 message only when a network delay is greater than the corresponding driver transition time. When false, PrimeTime displays the RC-009 message whenever it overrides the library-derived drive resistance during the RC-009 condition. This variable is effective only if the **rc_adjust_rd_when_less_than_rnet** variable is true.

The "RC-009 condition" means a condition in which PrimeTime checks the library-derived drive resistance, and if is less than the dynamic RC network impedance to ground by an amount equal to or greater than the value of the **rc_rd_less_than_rnet_threshold** variable, PrimeTime replaces the drive resistance with a value obtained by using an empirical formula to improve accuracy and issues the RC-009 message. The filtering provided by **rc_filter_rd_less_than_rnet** isolates those timing calculations known to be most sensitive to drive resistance. The network delay is not compared with the slew itself, but with with the time the driver reaches its later slew trip point. If you want the RC-009 message to be displayed whenever PrimeTime drive resistance to improve accuracy, set this variable to false.

This variable is one of a set of four variables related to the RC-009 condition. The others are as follows:

- **rc_adjust_rd_when_less_than_rnet** enables or disables the RC-009 condition; the default is true. When this variable is set to false, PrimeTime does not check the drive resistance, and the values of the other related variables do not matter.

- **rc_degrade_min_slew_when_rd_less_than_rnet** determines whether or not slew degradation is used in min analysis mode when the RC-009 condition occurs. The default is false. To use slew degradation, set this variable to true. This variable has no effect if **rc_adjust_rd_when_less_than_rnet** is false.

- **rc_rd_less_than_rnet_threshold** specifies the threshold beyond which PrimeTime overrides the library-derived drive resistance with an empirical formula. The default is 0.45 ohms. You can override this default by setting the variable to another value. This variable has no effect if **rc_adjust_rd_when_less_than_rnet** is false.

For more information, see the manual page of the RC-009 warning message.

To determine the current value of this variable, type **printvar rc_filter_rd_less_than_rnet** or **echo $rc_filter_rd_less_than_rnet**.

SEE ALSO

RC-009 (n).

rc_hide_ceff_warnings_for_enable_arcs

Specifies whether to hide warnings that occur during RC delay-calculation of C-effective for tri-state enable arcs.

TYPE

boolean

DEFAULT

false

DESCRIPTION

Some libraries use scalar models for tri-state enable arcs, so you can automatically suppress RC delay-calculation warnings for such arcs by setting **rc_hide_ceff_warnings_for_enable_arcs** true; otherwise, the **RC-004** warning message will be issued by default.

Please note that even when **rc_hide_ceff_warnings_for_enable_arcs** is true, tri-state enable arc results are still considered for propagation.

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

```
pt_shell> printvar rc_hide_ceff_warnings_for_enable_arcs
```

SEE ALSO

RC-004 (n).

rc_input_threshold_pct_fall

Specifies the percentage threshold of the startpoint to voltage source for a falling delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of 8 variables, listed in Table 1, that you must specify in order to perform delay calculation in the presence of annotated parasitics using the command **read_parasitics**. These variables interpret the cell delays and transition times from the Synopsys library.

We suggest the user to set the delay and slew trip point thresholds in the library directly, and suggest the user to keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

For the current value of this variable, type **printvar rc_input_threshold_pct_fall**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
      pt_shell> set rc_slew_lower_threshold_pct_fall 10
      pt_shell> set rc_slew_lower_threshold_pct_rise 10
      pt_shell> set rc_slew_upper_threshold_pct_fall 90
      pt_shell> set rc_slew_upper_threshold_pct_rise 90
      pt_shell> set rc_input_threshold_pct_rise 50
      pt_shell> set rc_input_threshold_pct_fall 50
      pt_shell> set rc_output_threshold_pct_rise 55
      pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (2), **report_annotated_parasitics** (2); **rc_input_threshold_pct_rise** (3), **rc_output_threshold_pct_rise** (3), **rc_output_threshold_pct_fall** (3), **rc_slew_lower_threshold_pct_rise** (3), **rc_slew_lower_threshold_pct_fall** (3), **rc_slew_upper_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_fall** (3).

rc_input_threshold_pct_rise

Specifies the percentage threshold of the startpoint to voltage source for a rising delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of 8 variables, listed in Table 1, that you must specify in order to perform delay calculation in the presence of annotated parasitics using the command **read_parasitics**. These variables interpret the cell delays and transition times from the Synopsys library.

We suggest the user to set the delay and slew trip point thresholds in the library directly, and suggest the user to keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_rise	**50.0**
rc_input_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

For the current value of this variable, type **printvar rc_input_threshold_pct_rise**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
pt_shell> set rc_slew_lower_threshold_pct_fall 10
pt_shell> set rc_slew_lower_threshold_pct_rise 10
pt_shell> set rc_slew_upper_threshold_pct_fall 90
pt_shell> set rc_slew_upper_threshold_pct_rise 90
pt_shell> set rc_input_threshold_pct_rise 50
pt_shell> set rc_input_threshold_pct_fall 50
pt_shell> set rc_output_threshold_pct_rise 55
pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (2), **report_annotated_parasitics** (2); **rc_input_threshold_pct_fall** (3), **rc_output_threshold_pct_fall** (3), **rc_output_threshold_pct_rise** (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).

rc_output_threshold_pct_fall

Specifies the percentage threshold of the endpoint to voltage source for a falling delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

Specifies the threshold voltage that defines the endpoint of the falling cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 - 100.0 inclusive; the default is 50.0.

This variable is one of eight variables, listed in Table 1, that you must specify to perform delay calculation in the presence of annotated parasitics using the **read_parasitics** command. These variables interpret the cell delays and transition times from the Synopsys library.

It is suggested that you set the delay and slew trip point thresholds in the library directly and you should keep them in the range of 10 to 90.

Table 1 Variables to Specify When Performing Delay Calculation

Variable Name	Default
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

For the current value of this variable, type the following:

```
printvar rc_output_threshold_pct_fall
```

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
      pt_shell> set rc_slew_lower_threshold_pct_fall 10
      pt_shell> set rc_slew_lower_threshold_pct_rise 10
      pt_shell> set rc_slew_upper_threshold_pct_fall 90
      pt_shell> set rc_slew_upper_threshold_pct_rise 90
      pt_shell> set rc_input_threshold_pct_rise 50
      pt_shell> set rc_input_threshold_pct_fall 50
      pt_shell> set rc_output_threshold_pct_rise 55
      pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (2), **report_annotated_parasitics** (2); **rc_input_threshold_pct_fall** (3), **rc_input_threshold_pct_rise** (3), **rc_output_threshold_pct_rise** (3), **rc_slew_lower_threshold_pct_rise** (3), **rc_slew_lower_threshold_pct_fall** (3), **rc_slew_upper_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_fall** (3).

rc_output_threshold_pct_rise

Specifies the percentage threshold of the endpoint to voltage source for a rising delay calculation.

TYPE

float

DEFAULT

50

DESCRIPTION

Specifies the threshold voltage that defines the endpoint of the rising cell or net delay calculation. The value is a percent of the voltage source. The allowed values are 0.0 - 100.0 inclusive and the default is 50.0.

This variable is one of eight variables, listed in Table 1, that you must specify to perform delay calculation in the presence of annotated parasitics using the **read_parasitics** command. These variables interpret the cell delays and transition times from the Synopsys library.

It is suggested that you set the delay and slew trip point thresholds in the library directly and that you keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

For the current value of this variable, type the following:

```
printvar rc_output_threshold_pct_rise
```

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
      pt_shell> set rc_slew_lower_threshold_pct_fall 10
      pt_shell> set rc_slew_lower_threshold_pct_rise 10
      pt_shell> set rc_slew_upper_threshold_pct_fall 90
      pt_shell> set rc_slew_upper_threshold_pct_rise 90
      pt_shell> set rc_input_threshold_pct_rise 50
      pt_shell> set rc_input_threshold_pct_fall 50
      pt_shell> set rc_output_threshold_pct_rise 55
      pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (2), **report_annotated_parasitics** (2), **rc_input_threshold_pct_fall** (3), **rc_input_threshold_pct_rise** (3), **rc_output_threshold_pct_fall** (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).

rc_rd_less_than_rnet_threshold

Specifies the RC-009 threshold, beyond which PrimeTime overrides the library-derived drive resistance with an empirical formula, to improve accuracy.

TYPE

double

DEFAULT

0.45

DESCRIPTION

This variable specifies a metric threshold to be used by PrimeTime to determine whether to adjust the library-derived drive resistance using an empirical formula; see the RC-009 man page for more information about this condition. This variable is effective only if the **rc_adjust_rd_when_less_than_rnet** variable is true. The default value of this variable is 0.45.

To determine the value appropriate for a given technology, look at PrimeTime accuracy vs. drive-strength for cells connected to very resistive networks, e.g. top-level routes, with **rc_adjust_rd_when_less_than_rnet** set false. At a particular drive-strength the accuracy will suffer due to the limitation in the delay/slew schema that RC-009 addresses. Then set **rc_adjust_rd_when_less_than_rnet** true and use a value of **rc_rd_less_than_rnet_threshold** high-enough to be used for all the drivers (e.g. 1.0). Next, gradually decrease **rc_rd_less_than_rnet_threshold** until RC-009 switches off at the desired drive-strength.

Customers who have performed this study have so far obtained values very close to the default value of 0.45. As technology shrinks, so will drive-resistances, causing an increased occurrance of RC-009; in that case users can decrease the **rc_rd_less_than_rnet_threshold** variable or switch to using Composite Current-Source data for delay calculation.

To determine the current value of this variable, type **printvar rc_less_than_rnet_threshold** or **echo $rc_less_than_rnet_threshold**.

SEE ALSO

rc_degrade_min_slew_when_rd_less_than_rnet (3), **rc_filter_rd_less_than_rnet** (3), **rc_rd_less_than_rnet_threshold** (3); **RC-009** (n).

rc_receiver_model_mode

Specifies which receiver model type to use for RC delay calculation.

TYPE

string

DEFAULT

advanced

DESCRIPTION

PrimeTime supports two types of receiver models for RC delay calculation, basic and advanced. The basic model is a single capacitance dependent only on the rise, fall, min, or max arc condition. The advanced model is a voltage-dependent capacitance additionally dependent on input-slew and output capacitance. The advanced model has many advantages, one of which is that the accuracy of **both** delays *and* slews is improved. Another advantage is that nonlinearities such as the Miller effect are addressed. The advanced receiver model is part of the Synopsys Composite Current-Source (CCS) model.

When set to *advanced*, RC delay calculation will use the advanced receiver model if data for it is present *and* if the network is driven by the advanced driver model. The **report_delay_calculation** command used on a network arc will show the message "Advanced receiver-modeling used" as appropriate.

When the shell variable **rc_receiver_model_mode** is set to *advanced*, and the network is not driven by the advanced driver model, (i.e. the variable **rc_driver_model_mode** is set to *basic* or lumped load is used), PrimeTime will use the advanced voltage-dependent capacitance models to derive an equivalent single capacitance dependent only on the rise, fall, min or max arc condition. These equivalent capacitances will be used in analysis instead of the pin capacitances from the library. The **report_delay_calculation** command used on a network arc will not show the message "Advanced receiver-modeling used" for these calculations, since only an equivalent single capacitance is used.

When the shell variable **rc_receiver_model_mode** is set to *basic*, RC delay calculation will always use the pin capacitances specified in the design libraries.

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

pt_shell> **printvar rc_receiver_model_mode**

SEE ALSO

rc_driver_model_mode (3), **report_delay_calculation** (2).

rc_slew_derate_from_library

Specifies the derating needed for the transition time in library to match that from the characterization trip points.

TYPE

float

DEFAULT

1

DESCRIPTION

A floating point number between 0.0 and 1.0 that specifies the derating needed for the transition times in the Synopsys library to match the transition times between the characterization trip points. The default is 1.0, meaning that the transition times in the Synopsys library are used without change. Set this variable only when using reduced or detailed parasitics annotation (for example, RSPF, DSPF, or SPEF files).

Usually, there is no need to set this variable, because as the transition times specified in the library represent the exact transition times between the characterization trip points. Use this variable for libraries in which the transition times of the Synopsys library are extrapolated to the rail voltages. For example, if the transition times are characterized between 30% and 70% and then extrapolated to the rails, **rc_slew_derate_from_library** should be set to 0.4 = (70 - 30) / 100. For the current value of this variable, type **printvar rc_slew_derate_from_library**.

EXAMPLES

The following commands specify the threshold levels that were used to characterize cell delays, as defined in the Synopsys library.

```
pt_shell> set rc_slew_derate_from_library 0.4
pt_shell> set rc_slew_lower_threshold_pct_fall 30
pt_shell> set rc_slew_lower_threshold_pct_rise 30
pt_shell> set rc_slew_upper_threshold_pct_fall 70
pt_shell> set rc_slew_upper_threshold_pct_rise 70
pt_shell> set rc_input_threshold_pct_rise 50
pt_shell> set rc_input_threshold_pct_fall 50
pt_shell> set rc_output_threshold_pct_rise 50
pt_shell> set rc_output_threshold_pct_fall 50
```

The four "slew threshold" variables specify that slews were characterized by measuring the transition times from 30 to 70 percent and from 70 to 30 percent of the supply voltage. The "slew derate" variable, which is set to 0.4, specifies that the transition times were extrapolated to the rail voltages (0 to 100 percent of the supply voltage); the range of 30 to 70 percent is a span of 40 percent of the supply voltage. The two "input threshold" and two "output threshold" variables specify that delays were calculated from trip points at 50 percent of the supply voltage.

SEE ALSO

read_parasitics (2), **report_annotated_parasitics** (2),
rc_slew_lower_threshold_pct_rise (3), **rc_slew_lower_threshold_pct_fall** (3),
rc_slew_upper_threshold_pct_rise (3), **rc_slew_upper_threshold_pct_fall** (3),
rc_input_threshold_pct_rise (3), **rc_input_threshold_pct_fall** (3),
rc_output_threshold_pct_rise (3), **rc_output_threshold_pct_fall** (3).

rc_slew_lower_threshold_pct_fall

Specifies the percentage threshold of the lower slew endpoint to the voltage source for a falling transition.

TYPE

float

DEFAULT

20

DESCRIPTION

Specifies the threshold voltage that defines the endpoint of the falling slew calculation. The value is a percent of the voltage source. The allowed values are 0.0 - 100.0 inclusive and the default is 20.0.

This variable is one of eight variables, listed in Table 1, that you must specify to perform delay calculation in the presence of annotated parasitics using the **read_parasitics** command. These variables interpret the cell delays and transition times from the Synopsys library.

It is suggested that you set the delay and slew trip point thresholds in the library directly and keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

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

```
printvar rc_slew_lower_threshold_pct_fall
```

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
      pt_shell> set rc_slew_lower_threshold_pct_fall 10
      pt_shell> set rc_slew_lower_threshold_pct_rise 10
      pt_shell> set rc_slew_upper_threshold_pct_fall 90
      pt_shell> set rc_slew_upper_threshold_pct_rise 90
      pt_shell> set rc_input_threshold_pct_rise 50
      pt_shell> set rc_input_threshold_pct_fall 50
      pt_shell> set rc_output_threshold_pct_rise 55
      pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (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_lower_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_fall** (3).

rc_slew_lower_threshold_pct_rise

Specifies the percentage threshold of the lower slew startpoint to the voltage source for a rising transition.

TYPE

float

DEFAULT

20

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the rising slew calculation. The value is a percent of the voltage source. The allowed values are 0.0 - 100.0 inclusive and the default is 20.0.

This variable is one of eight variables, listed in Table 1, that you must specify to perform delay calculation in the presence of annotated parasitics using the **read_parasitics** command. These variables interpret the cell delays and transition times from the Synopsys library.

It is suggested that you set the delay and slew trip point thresholds in the library directly and keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

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

```
printvar rc_slew_lower_threshold_pct_rise
```

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
pt_shell> set rc_slew_lower_threshold_pct_fall 10
pt_shell> set rc_slew_lower_threshold_pct_rise 10
pt_shell> set rc_slew_upper_threshold_pct_fall 90
pt_shell> set rc_slew_upper_threshold_pct_rise 90
pt_shell> set rc_input_threshold_pct_rise 50
pt_shell> set rc_input_threshold_pct_fall 50
pt_shell> set rc_output_threshold_pct_rise 55
pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (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_lower_threshold_pct_fall(3), rc_slew_upper_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_fall** (3).

rc_slew_upper_threshold_pct_fall

Specifies the percentage threshold of the slew startpoint to the voltage source for a falling transition.

TYPE

float

DEFAULT

80

DESCRIPTION

Specifies the threshold voltage that defines the startpoint of the falling slew calculation. The value is a percent of the voltage source. The allowed values are 0.0 - 100.0 inclusive and the default is 80.0.

This variable is one of eight variables, listed in Table 1, that you must specify to perform delay calculation in the presence of annotated parasitics using the **read_parasitics** command. These variables interpret the cell delays and transition times from the Synopsys library.

It is suggested that you set the delay and slew trip point thresholds in the library directly and keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

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

```
printvar rc_slew_upper_threshold_pct_fall
```

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
pt_shell> set rc_slew_lower_threshold_pct_fall 10
pt_shell> set rc_slew_lower_threshold_pct_rise 10
pt_shell> set rc_slew_upper_threshold_pct_fall 90
pt_shell> set rc_slew_upper_threshold_pct_rise 90
pt_shell> set rc_input_threshold_pct_rise 50
pt_shell> set rc_input_threshold_pct_fall 50
pt_shell> set rc_output_threshold_pct_rise 55
pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (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_lower_threshold_pct_fall** (3), **rc_slew_lower_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_rise** (3).

rc_slew_upper_threshold_pct_rise

Specifies the percentage threshold of the upper slew endpoint to the voltage source for a rising transition.

TYPE

float

DEFAULT

80

DESCRIPTION

Specifies the threshold voltage that defines the endpoint of the rising slew calculation. The value is a percent of the voltage source. The allowed values are 0.0 - 100.0 inclusive and the default is 80.0.

This variable is one of eight variables, listed in Table 1, that you must specify to perform delay calculation in the presence of annotated parasitics using the **read_parasitics** command. These variables interpret the cell delays and transition times from the Synopsys library.

It is suggested that you set the delay and slew trip point thresholds in the library directly and keep them in the range of 10 to 90.

Table 1

Variable Name	**Default**
rc_slew_lower_threshold_pct_rise	**20.0**
rc_slew_lower_threshold_pct_fall	**20.0**
rc_slew_upper_threshold_pct_rise	**80.0**
rc_slew_upper_threshold_pct_fall	**80.0**
rc_input_threshold_pct_fall	**50.0**
rc_input_threshold_pct_rise	**50.0**
rc_output_threshold_pct_fall	**50.0**
rc_output_threshold_pct_rise	**50.0**

The default values specify that a cell delay is defined from 50% of the voltage value for the input transition to 50% of the voltage value for the output transition. The default values also specify that a transition time, or slew, is defined from 20% to 80% of the voltage.

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

printvar rc_slew_upper_threshold_pct_rise

EXAMPLES

The following example specifies that cell delays from the Synopsys library are computed from 50% of the input transition to 55% of the output transition. In addition, the example specifies that transition times in the Synopsys library represent the delay from 10% to 90% of the voltage source.

```
      pt_shell> set rc_slew_lower_threshold_pct_fall 10
      pt_shell> set rc_slew_lower_threshold_pct_rise 10
      pt_shell> set rc_slew_upper_threshold_pct_fall 90
      pt_shell> set rc_slew_upper_threshold_pct_rise 90
      pt_shell> set rc_input_threshold_pct_rise 50
      pt_shell> set rc_input_threshold_pct_fall 50
      pt_shell> set rc_output_threshold_pct_rise 55
      pt_shell> set rc_output_threshold_pct_fall 55
```

SEE ALSO

read_parasitics (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_lower_threshold_pct_fall** (3), **rc_slew_lower_threshold_pct_rise** (3), **rc_slew_upper_threshold_pct_fall** (3).

read_parasitics_load_locations

Specifies that read parasitics should load locations information during the reading of a parasitics file.

TYPE

fIbooleanfP

DEFAULT

false

DESCRIPTION

When this variable is set to true, **read_parasitics** will load the locations of various nodes of nets, pins, and ports that are present in the parasitic file. The default is false, in which case locations infomation will not be loaded into PrimeTime.

The location data is stored using attributes. The attributes are set to the coordinate value directly from the parasitics files, and no interpretation or unit conversion is performed. The following attributes are available on pin and port objects.

*x_coordinate (float) * y_coordinate (float)

These attributes define a single (x, y) point. The following attributes are available on cell and net objects.

* x_coordinate_min (float) * x_coordinate_max (float) * y_coordinate_min (float) * y_coordinate_max (float)

These attributes define a bounding box around the cell or net. For cells, the bounding box is computed using all pins of the cell. For nets, the bounding box is computed using all net terminals (port and pins). If the parasitics file includes coordinates for intermediate modes, these will also be considered for the net's bounding box.

If location data has been loaded, it will be included in any parasitics files written out by PrimeTime. If you remove the parasitics from a net (using the **remove_annotated_parasitics** command, for instance), PrimeTime also deletes the location data.

SEE ALSO

read_parasitics (2).

report_default_significant_digits

The default number of significant digits used to display values in reports.

TYPE

fIintfP

DEFAULT

2

DESCRIPTION

The **report_default_significant_digits** variable sets the default number of significant digits for many reports. Allowed values are 0-13; the default is 2. Some report commands (for example, the **report_timing** command) have a **-significant_digits** option that overrides the value of this variable.

Not all reports respond to this variable. Check the man page of individual reports to determine whether they support this feature.

To determine the current value of this variable, type **printvar report_default_significant_digits** or **echo report_default_significant_digits**.

SEE ALSO

report_timing (2)

sdc_save_source_file_information

Enables or disables source file name and line number information of a subset of SDC commands related to the current design constraints PrimeTime context.

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

Setting this variable to *true* enables PrimeTime to preserve source location information, namely the source file name and line number. The scope of source location tracking only applies to the four timing exceptions commands:

- set_false_path - set_multicycle_path - set_max_delay/set_min_delay

It is important to note that the value of this variable is only allowed to be modified as long as no exceptions have been input. If at least one exception command has already been successfully input, a setting of this variable will result in an error (CMD-013) and the variable value will remain unchanged.

Note that source information is not available for any commands that were not input using source. (Tcl files sourced using the ’-f’ command line option are internally processed through the source command for the purposes of this feature.) Therefore, commands entered interactively at the shell prompt would not preserve nor print source location data. Also, commands input inside control structures such as if-statements, loops, or procedure calls are not tracked accurately.

This location data per exception command could be viewed using either report_exceptions or report_timing -exceptions.

To determine the current value of the **sdc_save_source_file_information** variable, type the following:

printvar sdc_save_source_file_information

or

echo $sdc_save_source_file_information.

EXAMPLES

```
pt_shell> report_exceptions

****************************************
Report : exceptions
```

```
Design : dma
Version: Z-2007.06-Beta3-DEV
Date   : Thu Apr  5 19:42:40 2007
****************************************

Reasons :    f  invalid start points
             t  invalid end points
             p  non-existent paths
             o  overridden paths

From                To                    Setup          Hold
------------------------------------------------------------------------
arbiter/lat_reg/CK  { arbiter/state_reg_3_/D arbiter/state_reg_2_/D }
                                          cycles=2       *
         [ location = /path/constraints.tcl:197 ]
```

and

```
pt_shell> report_timing -exceptions all

****************************************
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
        -exceptions all
Design : dma
Version: Z-2007.06-Beta3-DEV
Date   : Thu Apr  5 19:46:44 2007
****************************************

[ Timing report omitted. ]

The dominant exceptions are:
From          To             Setup                    Hold
------------------------------------------------------------------------

arbiter/lat_reg/CK
              { arbiter/state_reg_3_/D arbiter/state_reg_2_/D }
                             cycles=2                 *
         [ location = /path/constraints.tcl:197 ]

The overridden exceptions are:
        None
```

SEE ALSO

report_exceptions (2), **report_timing** (2).

sdc_version

Use in context of reading a Synopsys Design Constraints (SDC) file. Specifies the SDC version that was written.

TYPE

fIstringfP

DEFAULT

latest version

DESCRIPTION

The **sdc_version** variable is meaningful only within the context of reading an SDC file. Setting it outside an SDC file has no impact, other than to produce an informational message.

The **write_sdc** command writes a command to the SDC file to set the **sdc_version** variable to the version that was written. There is no user control over the version of SDC that is written. The most current version is written. When the **read_sdc** command reads the SDC file, it validates the version specified in the file (if present) with the version requested by the command.

SEE ALSO

read_sdc (2)
write_sdc (2)

sdc_write_unambiguous_names

Determines whether or not ambiguous hierarchical names are made unambiguous when they are written to SDC files.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true (the default), the application ensures that cell, net, pin, lib_cell, and lib_pin names written to the SDC file are not ambigious. When hierarchy has been partially flattened, embedded hierarchy separators can make names ambiguous, so that it is unclear which hierarchy separator characters are part of the name, and which are real separators. Beginning with SDC Version 1.2, hierarchical names can be made unambiguous using the **set_hierarchy_separator** SDC command and/or the **-hsc** option available on the **get_cells**, **get_lib_cells**, **get_lib_pins**, **get_nets**, and **get_pins** SDC object access commands. By default, PrimeTime and Design Compiler write an SDC file using these features to create unambiguous names.

The recommended practice is to accept the default behavior and allow the application to write SDC files that do not contain ambiguous names. However, if you are using a third-party application that does not support the unambiguous hierarchical names feature of SDC (in SDC Versions 1.2 and later), you can suppress this feature by setting the variable **sdc_write_unambiguous_names** to false. The **write_sdc** command issues a warning if you set this variable to false.

To determine the current value of this variable, use **printvar sdc_write_unambiguous_names**.

SEE ALSO

printvar (2), **write_sdc** (2).

sdf_align_multi_drive_cell_arcs

Boolean variable which specifies whether PrimeTime will unify the small timing differences in driver cell outputs of a parallel network, when writing out sdf delay values.

TYPE

boolean

DEFAULT

false

DESCRIPTION

Small timing differences in the timed switching characteristics of the mesh arcs can cause the simulation to fail. By setting *sdf_align_multi_drive_cell_arcs* to true, PrimeTime will attempt to align the delays between the driver pin(s) of the parallel network and the load pin(s) of the network. The cell and net delay arcs written to the sdf file will be adjusted to make this happen. The net arcs will only be altered if the variable **sdf_enable_port_construct** is set to true. Therefore,in order for the small timing differences to be eliminated, both **sdf_align_multi_drive_cell_arcs** and **sdf_enable_port_construct** must be set to true, and the following criteria must be true:

1. All cells have to be nonsequential, nonhierarchical cells.   2. All cells must be single input, single output devices.   3. None of the drivers of the parallel network are tristate buffers.

To determine the current value of this variable, use **printvar sdf_align_multi_drive_cell_arcs**.

SEE ALSO

sdf_align_multi_drive_cell_arcs_threshold (3), **sdf_enable_port_construct** (3), **sdf_enable_port_construct_threshold** (3).

sdf_align_multi_drive_cell_arcs_threshold

Specifies the threshold below which multi drive cell arcs will be aligned during *write_sdf*.

TYPE

float

DEFAULT

1 ps

DESCRIPTION

Small timing differences in the timed switching characteristics of the mesh arcs can cause the simulation to fail. By setting *sdf_align_multi_drive_cell_arcs* to true, PrimeTime will attempt to unify the delays between the driver pin(s) of the parallel network and the load pin(s) of the network. The cell delay arcs written to the sdf file will be adjusted to make this happen. The citeria for this to occur is that the delay values of the parallel cells are within a threshold of eachother, where the threshold is specidified by the variable *sdf_align_multi_drive_cell_arcs_threshold*. The threshold value is specified in pico seconds, where the default value is 1 ps.

To determine the current value of this variable, use *printvar sdf_align_multi_drive_cell_arcs_threshold*.

SEE ALSO

sdf_align_multi_drive_cell_arcs_threshold (3), **sdf_enable_port_construct** (3), **sdf_enable_port_construct_threshold** (3).

sdf_annotate_cond_specific_delays

Enables or disables support for out-of-order specific conditional delay annotations

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the value of this variable is set to *true*, when annotating conditional timing arcs, PT will give precedence to conditional SDF delay info over default delay info. This variable affects the read_sdf and set_annotated_delay commands.

SEE ALSO

read_sdf (2).

sdf_enable_cond_start_end

Enables or disables support for sdf_cond_start and sdf_cond_end attributes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the value of this variable is set to *true*, the variable enables PrimeTime to support sdf_cond_start and sdf_cond_end attributes on timing arcs, which affects the way the **read_sdf** command and **write_sdf** command deal with timing arcs.

SEE ALSO

read_sdf (2).

sdf_enable_port_construct

Enables or disables support for port construct usage during **write_sdf**.

TYPE

boolean

DEFAULT

false

DESCRIPTION

For designs with high-fanin, high-fanout mesh clock networks, large sdf files are produced. Setting the **sdf_enable_port_construct** variable to true will attempt to reduce the size of the produced sdf file. PrimeTime will use the port construct instead of the interconnect construct when executing a **write_sdf** command. The use of the port construct will be resticted by the variable **sdf_enable_port_construct_threshold**. Any group of parallel nets in the design, which are not driven or driving tristate buffers, and which have a delay value within a threshold as defined by the variable **sdf_enable_port_construct_threshold** will be written out using a port construct, otherwise the interconnect construct will be used. The port construct will not be used on nets outside the clock network. It must be noted that the produced sdf file can contain both port and interconnect statements for a given load pin. In this case, the port statement will be written out first, followed by interconnect statements.

SEE ALSO

sdf_enable_port_construct_threshold (3).

sdf_enable_port_construct_threshold

Sets the threshold value below within which the port construct during **write_sdf** will be used.

TYPE

float

DEFAULT

1 ps

DESCRIPTION

For designs with high-fanin, high-fanout mesh clock networks, large sdf files are produced. Setting the **sdf_enable_port_construct** variable to true will attempt to reduce the size of the produced sdf file. The **sdf_enable_port_construct_threshold** variable provides a maximum value for the parallel net arcs delay variance below which parallel nets will be written out using the port construct. The threshold value is specified in pico seconds, where the default value is 1 ps.

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

```
printvar sdf_enable_port_construct_threshold
```

or

echo $sdf_enable_port_construct_threshold.

SEE ALSO

sdf_enable_port_construct (3).

search_path

Shows a list of directory names that contain design and library files that are specified without directory names.

TYPE

fIlistfP

DEFAULT

"" (empty)

DESCRIPTION

A list of directory names that specifies which directories to search for design and library files that are specified without directory names. Normally, the **search_path** variable is set to a central library directory. The default value of this variable is the empty string, "". The **read_db** and **link_design** commands particularly depend on the **search_path** variable.

You can cause the **source** command to search for scripts using the **search_path** variable, by setting the **sh_source_uses_search_path** variable to *true*.

To determine the current value of this variable, type one of the following: **printvar search_path echo $search_path**

SEE ALSO

link_design (2)
printvar (2)
read_db (2)
sh_source_uses_search_path (3)
source (2)

sh_eco_enabled

Read-only variable that indicates if ECO commands are enabled.

TYPE

boolean

DEFAULT

false

DESCRIPTION

It indicates if the program is in ECO mode or not. In no-ECO mode, some commands (e.g., create_net) are not enabled in order to get better performance/capacity, and if those commands are called, they do nothing.

If it is required to change the value of this variable, please use set_program_options command.

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

```
pt_shell> printvar sh_eco_enabled
```

SEE ALSO

set_program_options (2)

sh_enable_line_editing

Enables the command line editing capabilities in PrimeTime.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If set to true it enables advanced unix like shell capabilities.

This variable needs to be set in .synopsys_pt.setup file to take effect.

Key Bindings

The **list_key_bindings** command displays current key bindings and the edit mode. To change the edit mode, variable **sh_line_editing_mode** can be set in either the .synopsys_pt.setup file or directly in the shell.

Command Completion

The editor will be able to complete commands, options, variables and files given a unique abbreviation. User need to type part of a word and hit the tab key to get the complete command, variable or file. For command options, users need to type ’-’ and hit tab key to get the options list.

If no match is found, the terminal bell rings. If the word is already complete a space is added to the end, if it isn’t already there, to speed typing and provide a visual indicator of successful completion. Completed text pushes the rest of the line to the right. If there are multiple matches then all the matching commands/ options/files or variables are autolisted.

Completion works in following context sensitive way :-

The first token of a command line : completes commands

Token that begins with "-" after a command : completes command arguments

After a ">", "|" or a "sh" command : completes filenames

After a set, unset or printvar command : completes the variables

After ’$’ symbol : completes the variables

After the help command : completes command

After the man command : completes commands or variables

Any token which is not the first token and doesn’t match any of the above rules : completes filenames

SEE ALSO

sh_line_editing_mode (3), **list_key_bindings** (2).

sh_fast_analysis_mode_enabled

Read-only variable that indicates if fast analysis mode is enabled.

TYPE

boolean

DEFAULT

false

DESCRIPTION

It indicates if the program is in fast analysis mode or not. In fast analysis mode, we do some tradeoff in accuracy for better performance.

If it is required to change the value of this variable, please use set_program_options command.

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

```
pt_shell> printvar sh_fast_analysis_mode_enabled
```

SEE ALSO

set_program_options (2)

sh_high_capacity_effort

Specifies the level of capacity effort for the timing analysis of PrimeTime/PrimeTime-SI. It only provides simple heuristics for tradeoff between capacity and performance of the program. It does not have any impact on the analysis results at all.

TYPE

string **sh_high_capacity_effort**

DEFAULT

default

DESCRIPTION

Specifies the effort level for capacity improved mode of the program. Allowed values are default, low, medium and high.

When effort level increases, the peak memory required by the tool is expected to reduce, with potentially slightly longer run time. It should be clarifed that this variable only provides simple heuristic control on the tradeoff between capacity and performance. And most importantly, regardless of the value, this variable alone does not change the results of the analysis.

This variable is only effective when the program is running in high capacity mode by issuing command **set_program_options -enable_high_capacity**.

If the program is already in high capacity mode, further change of this variable will not have any effect untill the next time the above command is issued.

To determine the current value of this variable, type **printvar sh_high_capacity_effort**.

SEE ALSO

set_program_options (2).

sh_high_capacity_enabled

A read-only variable for user to query whether high capacity mode is currently enabled or not. The value of this variable will change upon a successful change of the state with command **set_program_options**.

TYPE

boolean

DEFAULT

false for small designs, true for large designs

DESCRIPTION

This is a read-only variable. User can query its value for the mode of analysis and program accordingly. The value of this variable will change after successfully execute command **set_program_options**.

It is worth noting that **save_session** does not save the value of this variable, instead, the high capacity mode and this variable is inherited from the session where restore is done. Please refer to **set_program_options** for more details.

To determine the current value of this variable, type **printvar sh_high_capacity_enabled**.

SEE ALSO

set_program_options (1). **sh_high_capacity_effort** (3).

sh_launch_dir

Defines the launch directory of the current PrimeTime shell.

TYPE

string

DESCRIPTION

This read only variable defines the launch directory of the current PrimeTime shell. In multi-scenario analysis, all slaves are launched from the same directory as the master. However during the course of analysis the slave will change its current working directory multiple times however the /fBsh_launch_dir/fP variable remains constant accross all slaves and the master.

To determine the current value of this variable, type **printvar sh_launch_dir**

sh_limited_messages

The set of message types that have a limit by default in each invoking of read_parasitics, report_annotated_parasitics (with -check), read_sdf or update_timing. The limit is defined by **sh_message_limit**.

TYPE

string

DEFAULT

"DES-002 PARA-004 PARA-006 PARA-007 PARA-040 PARA-041 PARA-043 PARA-044 PARA-045 PARA-046 PARA-047 PARA-050 PARA-051 PARA-053 RC-002 RC-004 RC-005 RC-009 RC-011 RC-104 PTE-014 PTE-060 PTE-070"

DESCRIPTION

It defines the set of messages that have a limit by default when read_parasitics, report_annotated_parasitics (with -check option), read_sdf or update_timing is executed. This limit is not effective for messages emitted from other commands.

This limit is refreshed per invoking of read_parasitics, report_annotated_parasitics (with -check), read_sdf or update_timing (implicit or explicit). In other words, each invoking of read_parasitics, report_annotated_parasitics, read_sdf or update_timing allows **sh_message_limit** number of messages to show up for each message type in **sh_limited_messages**. If the limit is exceeded for one type of message, a warning message that shows this type of message will be suppressed shows up.

The setting of this variable has lower priority than command **set_message_info**. If **set_message_info** is already used to set the limit for a message type, the default limit on that message type is not effective.

If it is needed to remove this default limit, either set the **sh_limited_messages** to "" or set the **sh_message_limit** to 0.

To determine the current value of this variable, enter the following command:
pt_shell> **printvar sh_limited_messages**

SEE ALSO

sh_message_limit(3), set_message_info(2), get_message_info(2), print_message_info(2)

sh_line_editing_mode

Enables vi or emacs editing mode in PrimeTime shell.

TYPE

String

DEFAULT

emacs

DESCRIPTION

This variable can be used to set the command line editor mode to either vi or emacs. Valid values are emacs or vi.

Use **list_key_bindings** command to display the current key bindings and edit mode.

This variable can be set in the either .synopsys_pt.setup file or directly in the shell. The **sh_enable_line_editing** variable must be set to true.

SEE ALSO

sh_enable_line_editing (3), **list_key_bindings** (2).

sh_message_limit

Default limit of messages defined in **sh_limited_messages** during read_parasitics, report_annotated_parasitics (with -check), read_sdf and update_timing.

TYPE

int

DEFAULT

100

DESCRIPTION

It defines the default limit of the messages in sh_limited_messages when read_parasitics, report_annotated_parasitics (with -check option), read_sdf or update_timing is executed. This limit is not effective for messages emitted from other commands.

This limit is refreshed per invoking of read_parasitices, report_annotated_parasitics (with -check), read_sdf or update_timing (implicit or explicit). In other words, each invoking of read_parasitics, report_annotated_parasitics (with -check), read_sdf or update_timing allows **sh_message_limit** number of messages to show up for each message type in **sh_limited_messages**. If the limit is exceeded for one type of message, a warning message that shows this type of message will be suppressed shows up.

The setting of this variable has lower priority than command **set_message_info**. If **set_message_info** is already used to set the limit for a message type, the default limit on that message type is not effective.

If it is needed to remove this default limit, either set the **sh_limited_messages** to "" or set the **sh_message_limit** to 0.

To determine the current value of this variable, enter the following command:
pt_shell> **printvar sh_message_limit**

SEE ALSO

sh_limited_messages(3), set_message_info(2), get_message_info(2), print_message_info(2)

sh_output_log_file

Specifies the name of the file to which all application output is logged.

TYPE

string

DEFAULT

The empty string

DESCRIPTION

Specifies the name of the file to which the application logs all output during the session. By default, this variable is set to an empty string, indicating that the application's output is not logged.

This variable can be set only in a setup file. After setup files have been read, the variable becomes read- only.

To determine the current value of this variable, type **printvar sh_output_log_file**.

SEE ALSO

printvar (2), **sh_command_log_file** (3).

si_analysis_logical_correlation_mode

Enables or disables logical correlation analysis during PrimeTime-SI delay or noise calculation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true (the default), PrimeTime-SI enables logical correlation analysis while performing crosstalk delay or crosstalk noise analysis. In logical correlation analysis, PrimeTime-SI considers the logical relationships between multiple aggressor nets where buffers and inverters are used, so that the analysis is less pessimistic. When this variable is set to false, PrimeTime-SI assumes that the aggressor nets switch together in the direction that causes worst-case crosstalk delay or worst-case crosstalk noise bump on a victim net. When logical correlation analysis is turned off, PrimeTime-SI results are expected to be slightly more pessimistic but PrimeTime-SI runtime will be faster.

To determine the current value of this variable, type **printvar si_analysis_logical_correlation_mode**.

SEE ALSO

si_enable_analysis(3).

si_ccs_aggressor_alignment_mode

Specifies aggressor alignment mode used in the CCS-based gate-level simulation engine.

TYPE

String

DEFAULT

lookahead

DESCRIPTION

Specifies aggressor alignment mode used in the CCS-based gate-level simulation engine. Valid values include **stage** and **lookahead**. The default value is **lookahead** which enables the lookahead alignment feature for the CCS-based gate-level simulation engine so that PrimeTime-SI will find the alignment which results to worst-case path delay. It is noted that such alignment may not correspond to worst-case stage delay. The lookahead alignment feature only applies to the CCS-based gate-level simulation engine. Therefore, the variable **si_ccs_use_gate_level_simulation** must be set to true for the lookahead alignment mode to take effect.

For complete information about the difference between worst-case stage alignment and worst-case path alignment, see the PrimeTime-SI User Guide.

To determine the current value of this variable, type **printvar si_ccs_aggressor_alignment_mode**.

SEE ALSO

si_ccs_use_gate_level_simulation (3).

si_ccs_use_gate_level_simulation

Enables or disables the unified CCS timing and CCS noise engine for delay analysis.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true, enables the unified CCS timing and CCS noise engine to be used in delay analysis. By default, this variable is set to true. In order to use the unified CCS timing and CCS noise engine, in addition to setting this variable to true, you must also make sure that your library contains characterized CCS data.

For complete information about the unified CCS timing and CCS noise feature, see the PrimeTime-SI User Guide.

To determine the current value of this variable, type **printvar si_ccs_use_gate_level_simulation**.

SEE ALSO

update_timing (2), **report_timing** (2), **si_ccs_aggressor_alignment_mode** (3).

si_enable_analysis

Enables or disables PrimeTime-SI, which provides crosstalk analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When true, enables PrimeTime-SI, so that the crosstalk-aware timing calculation mode is used by **update_timing** and **report_timing**. By default, PrimeTime-Si is disabled; this variable is set to false.

If you set this variable to true and enable PrimeTime-SI, you must also do the following:

1. Obtain a PrimeTime-SI license. You cannot use PrimeTime-SI without a license.

2. Use **read_parasitics -keep_capacitive_coupling** to read in the coupling parasitics for your design. PrimeTime-SI is useful only if the design has coupling parasitics data.

For complete information about PrimeTime-SI, see the *PrimeTime Signal Integrity User Guide*.

To determine the current value of this variable, type **printvar si_enable_analysis**.

SEE ALSO

read_parasitics (2), **report_timing** (2), **update_timing** (2).

si_filter_accum_aggr_noise_peak_ratio

Specifies the threshold for the accumulated voltage bumps introduced by aggressors at a victim node, divided by Vcc, below which aggressor nets can be filtered out during electrical filtering.

TYPE

float

DEFAULT

0.03

DESCRIPTION

Specifies the threshold for the accumulated voltage bumps introduced by aggressors at a victim node; the default is 0.03. This variable, along with **si_filter_per_aggr_noise_peak_ratio**, makes up a pair of variables used by PrimeTime-SI during the electrical filtering phase, to determine whether an aggressor net can be filtered.

An aggressor net, along with its coupling capacitors, is filtered when either of the following are true:

1. The peak voltage of the voltage bump induced on the victim net divided by Vcc is less than the value of **si_filter_per_aggr_noise_peak_ratio**.

2. The accumulated peak voltage of voltage bumps induced on the victim by aggressor to the victim net divided by Vcc is less than the value of **si_filter_accum_aggr_noise_peak_ratio**.

To determine the current value of this variable, type **printvar si_filter_accum_aggr_noise_peak_ratio**.

SEE ALSO

si_analysis_effort_level (3), **si_enable_analysis** (3), **si_filter_per_aggr_noise_peak_ratio** (3), **si_filter_per_aggr_to_average_aggr_xcap_ratio** (3), **si_filter_per_aggr_xcap** (3), **si_filter_per_aggr_xcap_to_gcap_ratio** (3), **si_filter_total_aggr_xcap** (3), **si_filter_total_aggr_xcap_to_gcap_ratio** (3), **si_xtalk_reselect_delta_delay** (3), **si_xtalk_reselect_delta_delay_ratio** (3), **si_xtalk_reselect_max_mode_slack** (3), **si_xtalk_reselect_min_mode_slack** (3).

si_filter_per_aggr_noise_peak_ratio

Specifies the threshold for the voltage bump introduced by an aggressor at a victim node, divided by Vcc, below which the aggressor net can be filtered out during electrical filtering.

TYPE

float

DEFAULT

0.01

DESCRIPTION

Specifies the threshold for the voltage bump introduced by an aggressor at a victim node; the default is 0.01. This variable, along with **si_filter_accum_aggr_noise_peak_ratio**, makes up a pair of variables used by PrimeTime-SI during the electrical filtering phase, to determine whether an aggressor net can be filtered.

An aggressor net, along with its coupling capacitors, is filtered when either of the following are true:

1. The peak voltage of the voltage bump induced on the victim net divided by Vcc is less than the value of **si_filter_per_aggr_noise_peak_ratio**.

2. The accumulated peak voltage of voltage bumps induced on the victim by aggressors to the victim net divided by Vcc is less than the value of **si_filter_accum_aggr_noise_peak_ratio**.

To determine the current value of this variable, type **printvar si_filter_per_aggr_noise_peak_ratio**.

SEE ALSO

si_analysis_effort_level (3), **si_enable_analysis** (3), **si_filter_per_aggr_to_average_aggr_xcap_ratio** (3), **si_filter_per_aggr_xcap** (3), **si_filter_per_aggr_xcap_to_gcap_ratio** (3), **si_filter_accum_aggr_noise_peak_ratio** (3), **si_filter_total_aggr_xcap** (3), **si_filter_total_aggr_xcap_to_gcap_ratio** (3), **si_xtalk_reselect_delta_delay** (3), **si_xtalk_reselect_delta_delay_ratio** (3), **si_xtalk_reselect_max_mode_slack** (3), **si_xtalk_reselect_min_mode_slack** (3).

si_ilm_keep_si_user_excluded_aggressors

TYPE

fIbooleanfP

DEFAULT

FALSE

DESCRIPTION

Specifies whether user excluded SI aggressors are to be included in the SI-ILM or not. By default, no aggressors that are excluded by user are included in SI-ILM. Aggressors can excluded by setting nets to be constant or by using **set_si_delay_analysis** or **set_si_noise_analysis** commands.

To determine the current value of this variable, type **printvar si_ilm_keep_si_user_excluded_aggressors** or **echo $si_ilm_keep_si_user_excluded_aggressors**

SEE ALSO

create_ilm (2).

si_noise_composite_aggr_mode

Specifies the composite aggressor mode for noise analysis.

TYPE

String

DEFAULT

disabled

DESCRIPTION

This variable specifies which composite aggressor mode is used in PrimeTime SI noise analysis. Allowed values are *disabled* (the default), which turns off the composite aggressor feature. *normal*, causes PrimeTime SI to calculate noise by utilizing the non-statistical composite aggressor feature. Selecting *statistical* causes PrimeTime SI to calculate noise by using the statistical composite aggressor flow.

In *disabled* composite aggressor mode, PrimeTime SI uses its original flow with composite aggressor completely off to analyze the noise.

In *normal* composite aggressor mode, PrimeTime SI aggregates the effect of multiple small aggressors into a single composite aggressor, thereby reducing the computational complexity and improving the performance.

statistical composite aggressor mode reduces the pessimism for noise analysis by reducing the effect of composite aggressor.

For the current value of this variable, type **printvar si_noise_composite_aggr_mode**.

SEE ALSO

printvar (2), **report_noise_calculation** (2).

si_noise_endpoint_height_threshold_ratio

Specifies a value that defines the threshold where noise propagation stops. The ratio is between 0.0 and 1.0 of VDD.

TYPE

float

DEFAULT

0.75

DESCRIPTION

This variable sets a threshold voltage for an endpoint. When the propagated noise reaches this threshold voltage, noise propagation stops, and the load pin of the net is recorded as an endpoint.

This variable applies only to combinational circuit pins because sequential cell pins are automatically (noise) endpoints.

Suppose VDD is 1.0 V, and the variable is set to 0.75. In addition, suppose net N1 has a noise bump with the height of 0.8 V. Since the height of the noise bump is greater than 0.75 V, net N1 is recorded as an endpoint.

Since N1 is an endpoint, there is no propagated noise at the next stage of net N1.

To determine the current value of this variable, type **printvar si_noise_endpoint_height_threshold_ratio** or. **echo $si_noise_endpoint_height_threshold_ratio**.

SEE ALSO

report_noise (2), **report_noise_calculation** (2).

si_noise_limit_propagation_ratio

This variable limits the amount of propagated noise if the noise height passes noise immunity.

TYPE

float

DEFAULT

0.75

DESCRIPTION

During the noise update, if a noise passes the immunity criteria, then the propagated height is reduced to a specified ratio of the noise immunity value. This ratio is set by the variable **si_noise_limit_propagation_ratio**. This variable has to be between 0.0 and 1.0 and the default value is 0.75.

SEE ALSO

update_noise (2),

si_noise_slack_skip_disabled_arcs

Controls whether to skip disabled timing arcs for noise slack calculation.

TYPE

fIbooleanfP

DEFAULT

FALSE

DESCRIPTION

Controls whether to skip disabled timing arcs for noise slack calculation. Allowed values are *TRUE* or *FALSE* (the default).

When set to *TRUE*, noise slack is not calculated for disabled timing arcs.

When set to *FALSE*, disabled timing arcs are ignored and noise slacks are calculated for all available timing arcs.

For example, if an arc is disabled by set_case_analysis or set_disable_timing command, no noise slack is calculated for that arc by default. However, if the variable is set to *FALSE*, the disabled timing arc is ignored, and a noise slack is calculated.

To determine the current value of this variable, type **printvar si_noise_slack_skip_disabled_arcs** or **echo $si_noise_slack_skip_disabled_arcs**.

SEE ALSO

printvar (2), **get_timing_arcs** (2), **set_case_analysis** (2).

si_noise_update_status_level

Controls the number of progress messages displayed during the update of noise analysis.

TYPE

fIstringfP

DEFAULT

none

DESCRIPTION

Controls the number of progress messages displayed during the noise update process. Allowed values are *none* (the default), *low*, or *high*.

When set to *none*, no messages are displayed. When set to *low*, or *high*, the progress of the noise update is reported for an explicit update (using the **update_noise** command) or for an implicit update invoked by another command (for example, **report_noise**) that forces a noise update. The number of messages varies based on the value of the variable, as follows:

When set to *low*, messages are displayed only at the beginning and the end of the update.

When set to *high*, all messages for *low* are displayed; in addition, messages for the noise calculation step show the completion percentage in steps of 10 percents.

To determine the current value of this variable, type **printvar si_noise_update_status_level** or **echo $si_noise_update_status_level**.

SEE ALSO

printvar (2), **report_noise** (2), **update_noise** (2).

si_use_driving_cell_derate_for_delta_delay

Allows crosstalk delta delay for one net to be derated using the relevant derate factor for the cell driving that net.

TYPE

boolean

DEFAULT

FALSE

GROUP

si_variables

DESCRIPTION

When this variable is set to *true* the crosstalk delta delays for each net will be derated using the derate factors from the cell driving that net.

The relevant derate factor to be applied will adhere to the same precedence rules as the driving cell itself. For example, if no instance-specific derate factor was set on the driving cell then the hierarchical cell, the library cell and finally the global derate factors will be checked for a relevant derate factor.

To see what derate factors are to be applied to the net in question, first obtain the driving cell ($driving_cell) and use: pt_shell> **report_timing_derate [get_cells $driving_cell]**

If the command **report_timing** is invoked with the **-derate** option then the un-derated crosstalk delta delay will be reported as before. In addition the derate column will report the net derate factor used to derate the delta-free net delay.

To determine the current value of this variable, enter the following command: pt_shell> **printvar si_use_driving_cell_derate_for_delta_delay** or pt_shell> **echo $si_use_driving_cell_derate_for_delta_delay**

SEE ALSO

set_timing_derate (2), **report_timing_derate** (2), **report_timing** (2).

si_xtalk_composite_aggr_mode

Specifies the composite aggressor mode for crosstalk delay.

TYPE

String

DEFAULT

disabled

DESCRIPTION

This variable specifies which composite aggressor mode is used in PrimeTime SI crosstalk delay analysis. Allowed values are *disabled* (the default), which turns off the composite aggressor feature. *normal*, causes PrimeTime SI to calculate crosstalk delay by utilizing the non-statistical composite aggressor feature. Selecting *statistical* causes PrimeTime SI to calculate crosstalk by using the statistical composite aggressor flow.

In *disabled* composite aggressor mode, PrimeTime SI uses its original flow with composite aggressor completely off to calculate the xtalk delay.

In *normal* composite aggressor mode, PrimeTime SI aggregates the effect of some small aggressors (including filtered ones) into a single composite aggressor, thereby reducing the computational complexity and improving the performance.

statistical composite aggressor mode reduces the pessimism for xtalk delay analysis by reducing the effect of composite aggressor.

For the current value of this variable, type **printvar si_xtalk_composite_aggr_mode**.

SEE ALSO

printvar (2), **report_delay_calculation** (2). **si_xtalk_composite_aggr_noise_peak_ratio** (3). **si_xtalk_composite_aggr_quantile_high_pct** (3). **remove_si_delay_disable_statistical** (2). **set_si_delay_disable_statistical** (2). **report_si_delay_analysis** (2).

si_xtalk_composite_aggr_noise_peak_ratio

Used to control the composite aggressor selection for xtalk analysis.

TYPE

float

DEFAULT

0.01

DESCRIPTION

Specifies the threshold value in crosstalk bump to Vdd ratio, below which aggressors are selected into composite aggressor group. The default value is 0.01, which means all the aggressor nets with crosstalk bump to Vdd ratio less than 0.01 will be selected into composite aggressor group. It works together with other filtering thresholds **si_filter_per_aggr_noise_peak_ratio** and **si_filter_accum_aggr_noise_peak_ratio** to determine which aggressors can be selected into composite aggressor group.

To determine the current value of this variable, type **printvar si_xtalk_composite_aggr_noise_peak_ratio** at the PT shell prompt.

SEE ALSO

si_xtalk_composite_aggr_mode (3). **si_filter_per_aggr_noise_peak_ratio** (3). **si_filter_accum_aggr_noise_peak_ratio** (3). **si_xtalk_composite_aggr_quantile_high_pct** (3). **remove_si_delay_disable_statistical** (2). **set_si_delay_disable_statistical** (2). **report_si_delay_analysis** (2).

si_xtalk_composite_aggr_quantile_high_pct

Used to control the composite aggressor creation for statistical analysis.

TYPE

float

DEFAULT

99.73

DESCRIPTION

This variable is set to the desired probability in percentage format that any given real combined bump height will be less than or equal to the computed composite aggressor bump height. Given the desired probability, the resulting quantile value for the composite aggressor bump height will be calculated.

The default value of this variable is 99.73, which corresponds to a 3-sigma probability that the real bump height from any randomly-chosen combination of aggressors will be covered by the composite aggressor bump height.

To determine the current value of this variable, type **printvar si_xtalk_composite_aggr_quantile_high_pct** at the PT shell prompt.

SEE ALSO

si_xtalk_composite_aggr_mode (3). **si_xtalk_composite_aggr_noise_peak_ratio** (3). **remove_si_delay_disable_statistical** (2). **set_si_delay_disable_statistical** (2). **report_si_delay_analysis** (2).

si_xtalk_delay_analysis_mode

Specifies the arrival window alignment mode for crosstalk delay.

TYPE

String

DEFAULT

all_paths

DESCRIPTION

This variable specifies how the alignment between victim & aggressors is performed in crosstalk delay analysis PrimeTime SI. Allowed values are *all_paths* (the default), which causes PrimeTime SI to calculate crosstalk for all paths through the victim net. *worst_path*, causes PrimeTime SI to calculate crosstalk for all the worst paths(the earliest/latest path) through the victim net. Selecting *all_violating_paths* causes PrimeTime SI to calculate crosstalk for all worst paths and paths with the negative slack.

In *all_paths* alignment mode, PrimeTime SI considers the largest possible crosstalk delta delay for the given victim & aggressor arrival windows. This gurantees that all paths going through the victim net with different arrival times are conservative. This is default and tradational way PTSI calculated delta delay. The limitation of this approach is worst crosstalk delta delay applied to all paths including the worst path which causes slack of the design to be pessimistic. When the path based analysis is done on a path using **report_timing -pba_mode** the above pessimism is removed for the specific path. However it is too expensive to do path based analysis analysis on all the paths of the design.

In *worst_path* alignment mode, PrimeTime SI aligns aggressors for the the earlist/ latest paths on the victim, so only crosstalk affecting to these worst path is considered. So only the crosstalk affect that makes the slowest (earliest) path any slower(faster) is calculated. If the slowest/earliest path is a *set_false_path*, the true path is considerd. Considering the worst path instead of all paths, typically generates smaller delta delays and the worst paths and the design slack becomes less pessimistic. This approach makes sure that design slack & worst path are conservative.

There is a caveat to *worst_path* alignment mode, the crosstalk delay is applicable to worst path only, so the sub-critical path delay may be inaccurate. The side affect of this limitation *report_timing -nworst N*, N>1 could report paths with optimistic slacks. Also *report_constraint -all_violators -max_delay -min_delay* will report less number of violations than really exist on the design. Also as violating critical paths is fixed, the optimistic sub-critical paths will be critical and start violating. Also bottleneck commands like *report_timing_bottleneck* and *report_si_bottleneck* will be less effective.

For some design flows the sub-critical path optimism is less of an issue if the design meets the timing constraints, i.e. all endpoints in the design show positive slacks. However, when the design has not met the timing yet, getting conservative crosstalk deltas for all the violating paths (whose slack is negative) is essential for the fixing flow. The alignment mode *all_violating_paths* addresses this by aligning the aggressors for all the violating and the worst path through any pin in the design. This means, all paths with negative slacks and also all the critical paths through any pin in the design (even if the slack for that worst path is positive) have a conservative slack. This mode may show more pessimism on worst paths than the worst_path mode and also the runtime might get slightly higher than *worst_path*.

To reduce pessimism furthur, in the new modes *worst_path* and *all_violating_path*, another feature is enabled, where in the clock n/w seperate delay calculation is done for each clock. This is usefull when multiple clocks are propagated in clock n/w to the clock selection muxes.

For the current value of this variable, type **printvar si_xtalk_delay_analysis_mode**.

SEE ALSO

printvar (2), **si_enable_analysis** (2), **report_timing** (3).

si_xtalk_double_switching_mode

Controls the double switching detection during the PrimeTime-SI timing analysis.

TYPE

String

DEFAULT

"disabled"

DESCRIPTION

Double switching detection mode can have one of these three values, "disabled", "clock_network" or "full_design". When this mode is "disabled", the default mode, this double switching detection is not enabled. When this variable is enabled (set to "clock_network" or "full_design"), during **update_timing** PrimeTime-SI checks that whether crosstalk bump on the switching victim could cause the output to switch twice (and cause a pulse) instead of of desiered single signal propagation.

To detect the potential double switching in the clock network, which could cause the double clocking (where the clock could switch twice on a the sensitive edge) or false clocking (where the switching bump on the non-sensitive edge could actually latch the state), set this value to "clock_network".

To detect the potential double switching in the data path as well as clock path set this variable to "full_design". Double switching on a data path is less severe then double switching on the clock network.

The double switching detection needs CCSN library information on the victim load cell.

After the update_timing user could access these information in two ways 1) by command **report_si_double_switching** or 2) by the net attributes **si_has_double_switching** & **si_double_switching_slack**. Refer the man page of **report_si_double_switching** for the command details. The victim net attribute **si_has_double_switching** is true when ever there is a potential double switching on any of the load pins. The victim net attribute **si_double_switching_slack** has the bump slack, reducing the switching bump by that much amount could remove the double switching. If the victim net doesn’t cause double switching **si_double_switching_slack** will be "POSITIVE". If the victim net load pins doesn’t have CCS-Noise model information, the attribute will be reported as "INIFINITY".

The victim nets having the double switching is automatically reselected to higher iteration so that they could be reanalyzed with more accurate analysis.

The double switching happens when, the switching bump & transition time are large and fed into driver which strong enough amplify this. To avoid double switching either of them can be reduced.

SEE ALSO

si_enable_analysis(3). **report_si_double_switching**(3).

si_xtalk_exit_on_max_iteration_count

Specifies a maximum number of incremental timing iterations, after which PrimeTime-SI exits the analysis loop.

TYPE

integer

DEFAULT

2

DESCRIPTION

Specifies a maximum number of incremental timing iterations. PrimeTime-SI exits the analysis loop after performing this number of iterations.

The default value of this variable is 2, meaning that PrimeTime-SI exits the analysis loop after performing two iterations. You can override this default by setting the variable to another integer; the minimum allowed value is 1.

You may also manually exit the analysis loop by pressing Control-C to send an interrupt signal to the PrimeTime process. The interrupt is handled as the above exit criteria, at the end of the current iteration of the crosstalk analysis. You cannot interrupt iteration immediately without exiting PrimeTime.

To determine the current value of this variable, type **printvar si_xtalk_exit_on_max_iteration_count**.

si_xtalk_exit_on_max_iteration_count_incr

Specifies a maximum number of timing iterations following what-if change (such as size_cell) to the design, after which PrimeTime-SI exits the analysis loop.

TYPE

integer

DEFAULT

2

DESCRIPTION

Update_timing for signal integrity (SI) is done in iterative way. The number of iterations is controlled by variable **si_xtalk_exit_on_max_iteration_count**. The **si_xtalk_exit_on_max_iteration_count_incr** has same function but is used when update_timing can be done incrementally. Incremental SI timing is only done after minor changes, such as size_cell, insert_buffer, set_coupling_separation. Large number of changes or any other change result in full_update_timing.

SEE ALSO

si_xtalk_exit_on_max_iteration_count (3).

si_xtalk_reselect_clock_network

Determines whether or not PrimeTime-SI reselects clock network nets for subsequent delay calculations.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When true (the default), PrimeTime-SI reselects nets in the clock network for the next iteration of delay calculations. It can be enabled with following other reselection variables,

si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_max_mode_slack
si_xtalk_reselect_min_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_clock_network**.

SEE ALSO

si_xtalk_reselect_max_mode_slack (3), **si_xtalk_reselect_min_mode_slack** (3), **si_xtalk_reselect_delta_delay** (3), **si_xtalk_reselect_delta_delay_ratio** (3).

si_xtalk_reselect_delta_delay

Specifies the threshold of net delay change caused by crosstalk analysis, above which PrimeTime-SI reselects the net for subsequent delay calculations.

TYPE

float

DEFAULT

5

DESCRIPTION

This variable specifies a reselection threshold in terms of absolute delta delay. Nets that have at least one net arc with a crosstalk-annotated delta delay above this threshold are selected for the next iteration of PrimeTime-SI delay calculations. Note that delta delays are annotated on net arcs, but they capture the change of stage delay (cell plus net) because of crosstalk.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_max_mode_slack
si_xtalk_reselect_min_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_delta_delay**.

SEE ALSO

si_xtalk_reselect_delta_delay_ratio (3), **si_xtalk_reselect_max_mode_slack** (3), **si_xtalk_reselect_min_mode_slack** (3).

si_xtalk_reselect_delta_delay_ratio

Specifies the threshold of the ratio of net delay change caused by crosstalk analysis to the total stage delay, above which PrimeTime-SI reselects a net for subsequent delay calculations.

TYPE

float

DEFAULT

0.95

DESCRIPTION

This variable specifies a reselection threshold in terms of the delta delay ratio. Nets that have at least one net arc with a crosstalk-annotated delta delay, where the ratio of the annotated delta to the stage delay is above this threshold, are selected for the next iteration of PrimeTime-SI delay calculations.

If a net has multiple stage delays (because of a net fanout greater than one or multiple cell arcs), PrimeTime-SI considers the stage delta delay and stage delay that result in higher delta to stage delay ratio, thus making reselection conservative.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay
si_xtalk_reselect_max_mode_slack
si_xtalk_reselect_min_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_delta_delay_ratio**.

SEE ALSO

si_xtalk_reselect_delta_delay (3), **si_xtalk_reselect_max_mode_slack** (3), **si_xtalk_reselect_min_mode_slack** (3).

si_xtalk_reselect_max_mode_slack

Specifies the max mode pin slack threshold, below which PrimeTime-SI reselects a net for subsequent delay calculations.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the pin slack threshold in the max mode. Nets that have at least one pin with a max mode slack below this threshold are selected for the next iteration of PrimeTime-SI delay calculations. Max-mode pin slack is the slack of the worst max-mode (setup) path through the pin.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_min_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_max_mode_slack**.

SEE ALSO

si_xtalk_reselect_delta_delay (3), **si_xtalk_reselect_delta_delay_ratio** (3), **si_xtalk_reselect_min_mode_slack** (3).

si_xtalk_reselect_min_mode_slack

Specifies the min mode pin slack threshold, below which PrimeTime-SI reselects a net for subsequent delay calculations.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the pin slack threshold in the min mode. Nets that have at least one pin with a min mode slack below this threshold are selected for the next iteration of PrimeTime-SI delay calculations. Min-mode pin slack is the slack of the worst min-mode (hold) path through the pin.

This variable is one of a set of four variables that determine net reselection criteria. The other three variables are as follows:

si_xtalk_reselect_delta_delay
si_xtalk_reselect_delta_delay_ratio
si_xtalk_reselect_max_mode_slack

To determine the current value of this variable, type **printvar si_xtalk_reselect_min_mode_slack**.

SEE ALSO

si_xtalk_reselect_delta_delay (3), **si_xtalk_reselect_delta_delay_ratio** (3), **si_xtalk_reselect_max_mode_slack** (3).

si_xtalk_reselect_time_borrowing_path

Determines whether or not PrimeTime-SI reselects time borrowing path nets for subsequent delay calculations.

TYPE

Boolean

DEFAULT

TRUE

DESCRIPTION

Determines whether or not PrimeTime-SI reselects time borrowing path nets for subsequent delay calculations. When *true*, PrimeTime-SI reselects time borrowing path nets for the next iteration. It can be enabled with slack based reselection. It reselects only coupled nets that are not filtered, as with other reselection criteria. This variable is useful for designs that contain level-sensitive latches.

For slack based reselection PrimeTime-SI reselects all nets that directly or indirectly borrow time from nets that are reselected based on the **si_xtalk_reselect_max_mode_slack** and **si_xtalk_reselect_min_mode_slack** variables. This includes level-sensitive latch loops.

When *false*, reselection based on borrowing is disabled.

To determine the current value of this variable, type:

printvar si_xtalk_reselect_time_borrowing_path

SEE ALSO

si_xtalk_reselect_delta_delay (3), **si_xtalk_reselect_delta_delay_ratio** (3), **si_xtalk_reselect_max_mode_slack** (3), **si_xtalk_reselect_min_mode_slack** (3).

svr_enable_vpp

Enables or disables preprocessing of Verilog files by the Verilog preprocessor.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable is used only by the native Verilog reader. When true, before the Verilog reader reads a Verilog file, the Verilog preprocessor scans for and expands the Verilog preprocessor directives ‘define, ‘undef, ‘include, ‘ifdef, ‘else, and ‘endif. Intermediate files from the preprocessor are created in the directory referenced by the **pt_tmp_dir** variable. Also, the ‘include directive uses the **search_path** to find files.

Very few structural Verilog files use preprocessor directives. Set this variable to true only if your Verilog file contains directives that require the preprocessor. Without the preprocessor, the native Verilog reader does not recognize these directives.
To determine the current value of this variable, type **printvar svr_enable_vpp** or **echo $svr_enable_vpp**.

SEE ALSO

printvar (2), **read_verilog** (2); **pt_tmp_dir** (3), **search_path** (3).

svr_keep_unconnected_nets

Used only by the native Verilog reader to preserve or discard unconnected nets.

TYPE

fIBooleanfP

DEFAULT

true

DESCRIPTION

This variable is used only by the native Verilog reader. When *true* (the default), unconnected nets are preserved. When *false*, unconnected nets are discarded.

To determine the current value of this variable, type **printvar svr_keep_unconnected_nets** or **echo $svr_keep_unconnected_nets**.

SEE ALSO

printvar(2)
read_verilog(2)

timing_all_clocks_propagated

Determines whether or not all clocks are created as propagated clocks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When true, all clocks subsequently created by **create_clock** or **create_generated_clock** are created as propagated clocks. When false (the default), clocks are created as nonpropagated clocks.

By default, **create_clock** and **create_generated_clock** create only nonpropagated clocks. You can subsequently define some or all clocks to be propagated clocks using the **set_propagated_clock** command. However, if you set the **timing_all_clocks_propagated** variable to true, **create_clock** and **create_generated_clock** subsequently create only propagated clocks. Setting this variable to true or false affects only clocks created after the setting is changed. Clocks created before the setting is changed retain their original condition (propagated or non-propagated).

To determine the current value of this variable, type **printvar timing_all_clocks_propagated** or **echo $timing_all_clocks_propagated.**

SEE ALSO

create_clock (2), **create_generated_clock** (2), **printvar** (2), **set_propagated_clock** (2).

timing_allow_short_path_borrowing

Enable time borrowing through level sensitive latches for hold time checks.

TYPE

boolean

DEFAULT

false

DESCRIPTION

This boolean variable affects time borrowing for short paths (used for hold checks) at a level-sensitive latch. PrimeTime, by default, performs time borrowing only for long paths (used for setup checks).

The default model is a conservative model for short paths. It is valid even during power-up transient state.

An aggressive model is to allow borrowing for short paths. It is valid only during steady state.

Enable borrowing for short paths by setting variable **timing_allow_short_path_borrowing** to *true*.

To determine the current value of this variable, type **printvar timing_allow_short_path_borrowing**.

SEE ALSO

report_timing (2), **set_max_time_borrow** (2).

timing_aocvm_analysis_mode

Configure an AOCVM analysis.

TYPE

string

DEFAULT

""

GROUP

timing_variables

DESCRIPTION

When this variable is set to "", the default AOCVM analysis is performed. The default analysis is defined as follows.

Depth is used to index the random component of variation in an AOCVM derate table. Depth is defined as the number of cell (or net) delay timing arcs in a path from the path common point. Separate depth values are calculated for cells and nets. Separate depth values are calculated for launch and capture paths. Both clock and data networks objects are included in the depth computation. Random coefficients affect the depth computation; for more information see the **set_aocvm_coefficient** manpage.

Distance is used to index the systematic component of variation in an AOCVM derate table. Distance is defined as the length of the diagonal of the bounding box enclosing the cell (or net) delay timing arcs in a path from the path common point. Separate bounding boxes and distance values are calculated for cells and nets. Both clock and data networks objects are included in the bounding box. The cell at the path endpoint is included in the cell bounding box. Only nodes and terminals of the network along the net arc are included in the net bounding box.

The **timing_aocvm_analysis_mode** variable is used to configure an AOCVM analysis. Choose from the following analysis modes, which can be combined:

- *clock_network_only*
- *combined_launch_capture_depth*
- *single_path_metrics*

To configure an AOCVM analysis, specify the analysis mode(s) required in the **timing_aocvm_analysis_mode** variable. For example:

```
    pt_shell> set timing_aocvm_analysis_mode "clock_network_only"
```

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

```
      pt_shell> printvar timing_aocvm_analysis_mode
```

The effect that of each of the AOCVM analysis modes has on the default analysis is described below in detail:

clock_network_only When this option is not specified (default), AOCVM derating is applied throughout the design and constant (OCV) derating is ignored.

When this option is specified, AOCVM derating is applied to arc delays in the clock network only. Clock network AOCVM depth and distance metrics are calculated based on clock network topology only. The data network receives constant (OCV) derating, if constant derates have been annotated for data network objects; otherwise they are not derated.

In the *clock_network_only* delay timing arcs in the data network are excluded from depth and distance calculations. The cell at the path endpoint is still included in the cell bounding box.

combined_launch_capture_depth In this mode, launch and capture depths are not calcualted separately. The launch and capture paths are considered together and a combined depth is calculated for the entire path.

single_path_metrics In this mode, a single set of path metrics is calculated for both cell and net objects. Separate path metrics are *not* calculated for cells and nets in a path. This behaviour is backwardly compatible with the legacy Tcl-based "LOCV" solution.

Distance is measured by computing the diagonal of a bounding box around all of the cells in a path. Ports in the path and the common point are also included. This distance is used to lookup the systematic component of variation for both cells and nets.

Depth is measured considering only the cells in a path. This depth is used to lookup the random component of variation for both cells and nets.

SEE ALSO

set_aocvm_coefficient (2),
read_aocvm (2),
remove_aocvm (2),
report_aocvm (2),
get_timing_paths (2),
report_timing (2).

timing_aocvm_enable_analysis

Enable PrimeTime's graph-based AOCVM analysis.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When *false* (default) the graph-based aocvm timing update is not performed. A path-based aocvm analysis can be performed in this mode using the **-aocvm_path_based** option on the **report_timing** and **get_timing_paths** commands. In this mode constant timing derates specified using the **set_timing_derate command are required to pessimistically bound the analysis. Ideally, you should specify constant derates that do not clip the range of the path-based AOCVM derates to avoid optimism.**

When *true* the graph-based aocvm timing update is performed as part of the **update_timing** command. A path-based aocvm analysis can also be performed in this mode. In this mode constant timing derates are not required and, in fact, constant derates for *static delays* are ignored. Graph-based AOCVM derates computed during **update_timing** tightly bound the path-based AOCVM derates without clipping their range. Note that setting this variable to *true* will automatically switch the design into *on_chip_variation* analysis mode using the **set_operating_conditions** command.

SEE ALSO

set_operating_conditions (2),
read_aocvm (2),
report_aocvm (2),
get_timing_paths (2),
report_timing (2).

timing_bidirectional_pin_max_transition_checks

Determines the extent of max transition design rule checks on bidirectional pins.

TYPE

String

DEFAULT

both

DESCRIPTION

The variable timing_bidirectional_pin_max_transition_checks determines the extent of a max transition design rule check for bidirectional pins. The variable can be one of three values *{both, driver, load}* and the default is *both*. The value *both* specifies the driver and load to be checked, while *driver* only specifies the driver and *load* only specifies the load.

To determine the current value of this variable, use **printvar timing_bidirectional_pin_max_transition_checks**.

timing_check_defaults

Defines the default checks for the **check_timing** command.

TYPE

list

DEFAULT

generated_clocks generic latch_fanout loops no_clock no_input_delay
unconstrained_endpoints pulse_clock_non_pulse_clock_merge

GROUP

timing_variables

DESCRIPTION

Defines the default checks to be performed when the **check_timing** command is executed without any options. The same default checks are also performed if the **check_timing** command is used with **-include** or **-exclude** options. The default check list defined by this variable can be overriden by either redefining it before **check_timing** is executed or using the **-override_defaults** option of the **check_timing** command.

If an undefined check is specified while redefining this variable, a warning will be issued by the next execution of the **check_timing** command.

SEE ALSO

check_timing (2).

timing_clock_gating_propagate_enable

Allows the gating enable signal delay to propagate through the gating cell.

TYPE

int

DEFAULT

true

DESCRIPTION

When set to *true* (the default), PrimeTime allows the delay and slew from the data line of the gating check to propagate. When set to *false*, PrimeTime blocks the delay and slew from the data line of the gating check from propagating. Only the delay and slew from the clock line is propagated.

If the output goes to a clock pin of a latch, setting this variable to *false* produces the most desirable behavior.

If the output goes to a data pin, setting this variable to *true* produces the most desirable behavior.

To determine the current value of this variable, type **printvar timing_clock_gating_propagate_enable** or **echo $timing_clock_gating_propagate_enable**.

SEE ALSO

timing_clock_reconvergence_pessimism

Select signal transition sense matching for computing clock reconvergence pessimism removal.

TYPE

string

DEFAULT

normal

DESCRIPTION

Determines how the value of the clock reconvergence pessimism removal (crpr) is computed with respect to transition sense. Allowed values are *normal* (the default) and *same_transition*.

When set to *normal*, the crpr value is computed even if the clock transitions to the source and destination latches are in different directions on the common clock path. It is computed separately for rise and fall transitions and the value with smaller absolute value is used.

When set to *same_transition*, the crpr value is computed only when the clock transition to the source and destination latches have a common path and the transition is in the same direction on each pin of the common path. Thus if the source and destination latches are triggered by different edge types, crpr is computed at the last common pin at which the launch and capture edges match.

If the variable is set to *same_transition* then the CRP for all min pulse width checks will be zero as they are calculated using different (i.e. rise and fall) clock edges.

To determine the current value of this variable, type **printvar timing_clock_reconvergence_pessimism** or **echo $timing_clock_reconvergence_pessimism**.

SEE ALSO

report_timing (2), **get_timing_path** (2). **timing_remove_clock_reconvergence_pessimism** (3).

timing_clock_source_driver_pin_use_driver_arc_compatibility

Select between backward compatible behavior of forcing an ideal ramp at driver pin primary clock sources or using the backward cell arc to compute a realistic driver model.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When *true*, PrimeTime asserts a zero-transition, ideal ramp model at driver pin (i.e. a cell output pin driving the interconnect network) for the purposes of computing the interconnect delay and target transition times at the corresponding load pins. When *false*, PrimeTime would use the backward cell arcs, provided at least one does exist, to compute a worst-case driver model. It is important to note that this behavioral change applies only to primary clock sources. That is clock sources of create_clock commands, and not create_generated_clock commands.

To determine the current value of this variable, type **printvar timing_clock_source_driver_pin_use_driver_arc_compatibility**.

SEE ALSO

printvar (2), **create_clock** (2).

timing_crpr_enable_adaptive_engine

Enables or disables the adaptive CRPR engine.

TYPE

Boolean

DEFAULT

False

GROUP

timing_variables

DESCRIPTION

When set to *TRUE*, this variable turns on the adaptive CRPR engine. The adaptive CRPR engine can significantly improve the performance and capacity of a timing update with SI enabled (variable **si_enable_analysis** = *TRUE*) across multiple iterations.

The performance and capacity gain is achieved by only calculating CRP for the violating portion of the design thus reducing the complexity of the CRPR analysis.

This variable is only applicable to SI analysis (**si_enable_analysis** is set to *TRUE*) and the max iteration count (**si_xtalk_exit_on_max_iteration_count**) is set to *2* or greater. If neither of the previous two conditions is TRUE then PrimeTime will revert to using the standard CRP calculation method. Two SI iterations are required because adaptive CRPR requires two interations to perform its analysis as opposed to standard CRPR which only requires one SI iteration.

It should be noted that the user may see differences in SI delta delays and hence slack values between adaptive and standard CRPR. This is because SI net reselection is dependent on slack values which in turn are dependent on CRPR. Since adaptive only calculates CRP during the second iteration slack-based net reselection during the first iteration will not be CRPR-aware. The result of this will be that more nets will be reselected after the first SI if adaptive CRPR is enabled. Since more nets are reselected after the first iteration a more accurate SI analysis (that is, less pessimistic) can be expected.

If the user wishes to compare standard and adaptive CRPR then the best approach would be to ’force’ the reselection of all nets using the command **set_si_delay_analysis**. Slacks between both runs may then be compared.

Use the following command to determine the current value of the variable:

```
   prompt> printvar timing_crpr_enable_adaptive_engine
```

SEE ALSO

```
update_timing (2),
si_enable_analysis (3),
si_xtalk_exit_on_max_iteration_count (3),
set_si_delay_analysis (2),
timing_crpr_threshold_ps (3),
timing_remove_clock_reconvergence_pessimism (3).
```

timing_crpr_minimize_grouping

Enables or disables the aggressive grouping during CRPR calculations..

TYPE

Boolean

DEFAULT

False

GROUP

timing_variables

DESCRIPTION

When set to *TRUE*, this variable reduces the amount of grouping performed during CRPR calculations. This is done to maximize the computed value of clock reconvergence pessimism. Setting this variable to *TRUE* can result in significant memory overhead as PT would not perform some optimizations during CRPR calculations. It should only be turned ON only if the difference in CRP value reported by report_timing and report_crpr differs by more than the CRPR threshold.

Use the following command to determine the current value of the variable:

```
   prompt> printvar timing_crpr_minimize_grouping
```

SEE ALSO

update_timing (2),
timing_crpr_threshold_ps (3),
timing_remove_clock_reconvergence_pessimism (3).

timing_crpr_remove_clock_to_data_crp

Allows the removal of Clock Reconvergence Pessimism (CRP) from paths that fan out directly from clock source to the data pins of sequential devices.

TYPE

boolean

DEFAULT

FALSE

DESCRIPTION

When this variable is set to *true* then CRP will be removed for all paths that fan out directly from clock source pins to the data pins of sequential devices.

When this variable is set to *false*, only the CRP up to the clock source pin which fans out to the data pin of the sequential device would be removed. This is because the path up to a clock source pin is considered to be a clock path. Consider the following example, where GCLK1 as a generated clock with CLK as its master clock. In this case, the CRP between pins A and B would removed irrespective of the value of the variable. However, when the variable is set to *true*, additional CRP between pins B and C would also be removed.

```
                                                +-- buff3 (D) --- FF1/D
                                                |
  A (CLK) --- buff1 (B GCLK1) --- buff2 (C) --- |-- buff4 (E) --- FF1/CP
```

It should be noted that when this variable is set to *true* all sequential devices that reside in the fanout of clock source pins must be handled seperately in the subsequent timing update. This may cause a severe performance degradation to the timing update.

```
      pt_shell> echo $timing_crpr_remove_clock_to_data_crp
      or
            pt_shell> printvar timing_crpr_remove_clock_to_data_crp
```

SEE ALSO

timing_remove_clock_reconvergence_pessimism (3).

timing_crpr_remove_muxed_clock_crp

Allow CRPR to consider common path reconvergence between related clocks.

TYPE

Boolean

DEFAULT

TRUE

GROUP

timing_variables

DESCRIPTION

This variable controls the CRPR in cases where two related clocks reconverge in the logic. Two clocks are related if one is a generated clock and the other is its parent, or both are generated clocks of the same parent clock. Although this variable name refers specifically to multiplexers, the variable applies to any situation where two related clocks reconverge within combinational logic.

If this variable is set to TRUE then the separate clock paths up to the multiplexer are treated as reconvergent, and the CRP will include the reconvergence point as well as any downstream common logic. If this variable is set to FALSE then the common pin will be the last point where the clocks diverged to become related clocks.

If the design contains related clocks which switch dynamically (a timing path launches from one related clock and the clock steering logic switches dynamically so the path captures on the other related clock), then this variable should be set to false so the CRP is not removed.

The default value is true, which removes the additional CRP.

To determine the value of this variable use:

```
   prompt> printvar timing_crpr_remove_muxed_clock_crp
```

SEE ALSO

timing_remove_clock_reconvergence_pessimism (3).

timing_crpr_threshold_ps

Specifies amount of pessimism that clock reconvergence pessimism removal (CRPR) is allowed to leave in the report.

TYPE

float

DEFAULT

20

DESCRIPTION

Specifies amount of pessimism that clock reconvergence pessimism removal (CRPR) is allowed to leave in the report. The unit is in pico seconds (ps), regardless of the units of the main library.

The threshold is per reported slack: setting the this variable to the *TH1* value means that reported slack is no worse than *S - TH1*, where *S* is the reported slack when **timing_crpr_threshold_ps** is set close to zero (the minimum allowed value is **1** picosecond).

The variable has no effect if CRPR is not active (**timing_remove_clock_reconvergence_pessimism** is false). The larger the value of **timing_crpr_threshold_ps**, the faster the runtime when CRPR is active. The recommended setting is about one half of the stage (gate plus net) delay of a typical stage in the clock network. It provides a reasonable trade-off between accuracy and runtime in most cases. You may want to use different settings throughout the design cycle: larger during the design phase, smaller for sign-off. You might have to experiment and set a different value when moving to a different technology.

To determine the current value of this variable, type **printvar timing_crpr_threshold_ps**.

SEE ALSO

timing_remove_clock_reconvergence_pessimism (3).

timing_disable_bus_contention_check

Disable checking for timing violations resulting from transient contention on design busses.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Applies only to bus designs that have multiple three-state drivers.

When *true*, PrimeTime ignores timing setup and hold (max and min) violations that occur as a result of transient bus contention. When *false* (the default), PrimeTime reports these timing violations.

Bus contention occurs when more than one driver is enabled at the same time. By default, PrimeTime treats the bus as if it is in an unknown state during this region of contention, and reports a timing violation if the setup and hold regions extend into the contention region. Note that checking is done only for timing violations, and not for logical and excessive power dissipation violations, which are outside the scope of static timing analysis tools.

Set this variable to *true* only if you are certain that transient bus contention regions will never occur. By setting the value to *true*, you guarantee that on a multi-driven three-state bus, the drivers in the previous clock cycle are disabled before the drivers in the current clock cycle are enabled. If you set this variable to *true*, you must ensure that the variable **timing_disable_bus_contention_check** is *false*. The variables **timing_disable_bus_contention_check** and **timing_disable_floating_bus_check** cannot both be *true* at the same time.

During the switching between the high-impedance (Z) state and the high/low state, the timing behavior (for example, intrinsic delay) of three-state buffers is captured in the Synopsys library using the timing arc types three_state_disable and three_state_enable. These timing arcs connect the enable pin to the output pin of the three-state buffers. For details, see the *Library Compiler Reference Manual*.

To determine the current value of this variable, type **printvar timing_disable_bus_contention_checks** or **echo $timing_disable_bus_contention_checks**.

SEE ALSO

timing_disable_floating_bus_check (3).

timing_disable_clock_gating_checks

Disable checking for setup and hold clock gating violations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, disables clock-gating setup and hold checks. When *false* (the default), PrimeTime automatically determines clock-gating and performs clock-gating setup and hold checks.

To determine the current value of this variable, type **printvar timing_disable_clock_gating_checks** or **echo $timing_disable_clock_gating_checks**.

SEE ALSO

report_constraint (2), **set_clock_gating_check** (2).

timing_disable_cond_default_arcs

Disable the default, non-conditional timing arc between pins that do have conditional arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, disables nonconditional timing arcs between any pair of pins that have at least one conditional arc. When *false* (the default), these nonconditional timing arcs are not disabled. This variable is primarily intended to deal with the situation between two pins that have conditional arcs, where there is always a default timing arc with no condition.

Set this variable to *true* when the specified conditions cover all possible state-dependent delays, so that the default arc is useless. For example, consider a 2-input XOR gate with inputs as A and B and with output as Z. If the delays between A and Z are specified with 2 arcs with respective conditions ’B’ and ’B~", the default arc between A and Z is useless and should be disabled.

To determine the current value of this variable, type **printvar timing_disable_cond_default_arcs** or **echo $timing_disable_cond_default_arcs**.

SEE ALSO

report_disable_timing (2).

timing_disable_floating_bus_check

Disable checking for timing violations resulting from transient floating design buses.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Applies only to bus designs that have multiple three-state drivers.

When *true*, PrimeTime ignores timing setup and hold (max and min) violations that occur as a result of transient floating buses. When *false* (the default), PrimeTime reports these timing violations.

Floating bus condition occurs when no driver controls the bus at a given time. By default, PrimeTime treats the bus as if it is in an unknown state during this region of contention, and reports a timing violation if the setup and hold regions extend into the floating region. Note that checking is done only for timing violations, and not for logical violations, which are outside the scope of static timing analysis tools.

Set this value to *true* only if you are certain that transient floating bus regions will never occur. By setting the value to *true*, you guarantee that on a multi-driven three-state bus, the drivers in the previous clock cycle are disabled before the new drivers in the current clock cycle are enabled. If you set this variable to *true*, you must ensure that the variable **timing_disable_bus_contention_check** is *false*. The variables **timing_disable_floating_bus_check** and **timing_disable_bus_contention_check** cannot both be *true* at the same time.

During the switching between the high-impedance (Z) state and the high/low state, the timing behavior (for example, intrinsic delay) of three-state buffers is captured in the Synopsys library using the timing arc types three_state_disable and three_state_enable. These timing arcs connect the enable pin to the output pin of the three-state buffers. For details, see the *Library Compiler Reference Manual*.

To determine the current value of this variable, type **printvar timing_disable_floating_bus_check** or **echo $timing_disable_floating_bus_check**.

SEE ALSO

timing_disable_bus_contention_check (3).

timing_disable_internal_inout_cell_paths

Enable bidirectional feedback paths within a cell.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When *true* (the default), PrimeTime automatically disables bidirectional feedback paths in a cell. When *false*, bidirectional feedback paths in cells are enabled.

This variable has no effect on timing of bidirectional feedback paths that involve more than one cell (that is, if nets are involved); these feedback paths are controlled by the variable **timing_disable_internal_inout_net_arcs**.

To determine the current value of this variable, type **printvar timing_disable_internal_inout_cell_paths** or **echo $timing_disable_internal_inout_cell_paths**.

SEE ALSO

remove_disable_timing (2), **report_timing** (2), **set_disable_timing** (2); **timing_disable_internal_inout_net_arcs** (3).

timing_disable_internal_inout_net_arcs

Controls whether bidirectional feedback paths across nets are disabled or not.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When *true* (the default), PrimeTime automatically disables bidirectional feedback paths that involve more than one cell; no path segmentation is required. Note that only the feedback net arc between non-bidirectional driver and load is disabled. When *false*, these bidirectional feedback paths are enabled.

This variable has no effect on timing of bidirectional feedback paths that are completely contained in one cell (that is, if nets are not involved); these feedback paths are controlled by the variable **timing_disable_internal_inout_cell_paths**.

To determine the current value of this variable, type **printvar timing_disable_internal_inout_net_arcs** or **echo $timing_disable_internal_inout_net_arcs**.

SEE ALSO

remove_disable_timing (2), **report_timing** (2), **set_disable_timing** (2); **timing_disable_internal_inout_cell_paths** (3).

timing_disable_recovery_removal_checks

Disable or enable the timing analysis of recovery and removal checks in the design.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, disables recovery and removal timing analysis. When *false* (the default), PrimeTime performs recovery and removal checks; for descriptions of these checks, see the man page for the **report_constraint** command.

To determine the current value of this variable, type **printvar timing_disable_recovery_removal_checks** or **echo $timing_disable_recovery_removal_checks**.

SEE ALSO

report_constraint (2).

timing_dynamic_loop_breaking

Enable or disable the dynamic breaking of combinational feedback loops.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, enables dynamic loop breaking. When *false* (the default), dynamic loop breaking is disabled.

By default, PrimeTime handles loops by identifying the loop and disabling one of the timing arcs of the loop. In some cases, this approach can result in some real paths not being reported in PrimeTime, because they are broken by the disabled arcs used to break loops. Enabling dynamic loop breaking guarantees that no timing arc is disabled to break a loop and that all valid paths of the design are reported.

If the design or the search space for report_timing is large, or if the loops are complex, setting this variable may increase the run time (or memory) for report_timing significantly.

To determine the current value of this variable, type **printvar timing_dynamic_loop_breaking**.

SEE ALSO

printvar (2), **report_timing** (2).

timing_early_launch_at_borrowing_latches

Removes clock latency pessimism from the launch times for paths which begin at the data pins of transparent latches.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

In the following description we assume that the data paths of interest are setup paths since we refer specifically to time borrowing scenarios. However, if **timing_allow_short_path_borrowing** is enabled then the same discussion applies to borrowing hold paths too.

When a latch is in its transparent phase, data arriving at the D-pin passes through the element as though it were combinational. To model this scenario, whenever PrimeTime determines that time borrowing occurs at such a D-pin, paths which originate at the D-pin are created.

Sometimes there is a difference between the launching and capturing latch latencies, due either to reconvergent paths in the clock network or different min and max delays of cells in the clock network. For setup paths, PrimeTime uses the late value to launch and the early value to capture. This achieves the tightest constraint and avoids optimism. However, for paths starting from latch D-pins this is pessimistic since data simply passes through and thus does not even "see" the clock edge at the latch.

When this timing variable is set to *true* (the default), such pessimism is eliminated by using the early latch latency to launch such paths. Note that only paths which originate from a latch D-pin are affected. When the variable is set to *false*, late clock latency is used to launch all setup paths in the design.

It is recommended that the user avail of this form of pessimism removal since it does not cause the run-time of the analysis to increase. However, it is also advised that the user disable it when clock reconvergence pessimism removal (CRPR) is enabled (i.e. when **timing_remove_clock_reconvergence_pessimism** is *true*). CRPR may not be applied to paths which have been launched using an early latency or the results may be optimistic. Since CRPR is a more sophisticated and accurate means of pessimism removal, the user should disable **timing_early_launch_at_borrowing_latches** when CRPR is enabled so that CRPR applies to all paths in the design. In this mode, note that the D-pin launch time is not modified by the open edge CRP - since late launch latency is used at the path startpoint, to additionally add CRP would be pessimistic, representing a "double-counting" of early-late differences.

To determine the current value of this variable, type **printvar timing_early_launch_at_borrowing_latches** or **echo**

$timing_early_launch_at_borrowing_latches.

SEE ALSO

printvar (2), **report_timing** (2), **timing_remove_clock_reconvergence_pessimism** (3), **timing_allow_short_path_borrowing** (3).

timing_edge_specific_source_latency

Controls whether the generated clock source latency computation will consider edge relationship or not.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When *true* (the default), only the paths with the same sense relationship derived from generated clock definition will be considered. When *false*, all paths fanout to generated clock source pin will be considered and the worst path will be selected for generated clock source latency computation.

For the current value of this variable, type **printvar timing_edge_specific_source_latency**.

SEE ALSO

create_generated_clock (2).

timing_enable_clock_propagation_through_preset_clear

Enables propagation of clock signals through preset and clear pins

TYPE

boolean

DEFAULT

false

GROUP

Timing variables

DESCRIPTION

When this variable is set to *true*, clock signals will be propagated through the preset and clear pins of a sequential device. Naturally, this will only occur when clock signals are incident on such pins.

If CRPR is enabled it will consider any sequential devices in the fanout of such pins for analysis.

Use the following command to determine the current value of the variable:

```
   prompt> printvar timing_enable_clock_propagation_through_preset_clear
```

SEE ALSO

timing_remove_clock_reconvergence_pessimism (2).

timing_enable_clock_propagation_through_three_state_enable_pins

Allow the clocks to propagate through the enable pin of a three-state cell.

TYPE

int

DEFAULT

false

DESCRIPTION

When *true*, PrimeTime allows the clocks to propagated through the enable pins of tristates. When *false* (the default), PrimeTime will not propagate clocks between a pair of pins if there is at least one timing arc with a disable sense between those pins.

To determine the current value of this variable, type **printvar timing_enable_clock_propagation_through_three_state_enable_pins** or **echo $timing_enable_clock_propagation_through_three_state_enable_pins**.

timing_enable_constraint_delay_calculation_compatibility

Indicates whether to revert to constraint arc delay calculation behavior to that or an earlier release of PrimeTime.

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

Indicates whether to revert to constraint arc delay calculation behavior prior to PrimeTime version W-2004.12. The change in behavior improves the delay calculation handling of constraint arcs under different operating conditions and run scenarios.

Prior to W-2004.12, PrimeTime would compute the constraint delay between a clock pin and a data pin in the following manner. The constraint value is equal to the worst delay between the library arc lookups for the min and max transitions at the clock pin. If the user specified different libraries for min and max conditions under on-chip variations mode, then the total calculations becomes four. In this case, however, PrimeTime would always select the max transition at the data pin for setup constraints and the min transition for hold.

As of W-2004.12, the latter limitation at the data pin is removed. Thus, with worst slew propagation mode and on-chip variations analysis with different min and max libraries, PrimeTime would maximize over eight computations: minimum/maximum transition at the data pin, minimum/maximum transition at the clock pin, and minimum/maximum library lookup tables.

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

printvar timing_enable_constraint_delay_calculation_compatibility
echo $timing_enable_constraint_delay_calculation_compatibility

SEE ALSO

set_operating_conditions (1)
printvar (2)

timing_enable_invalid_slew_propagation_compatibility

Enable or disable propagation of slew values driven by disabled arcs or non-existent transitions of half-unate arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, PrimeTime does propagate a slew from disabled arcs and non-existent transitions of half-unate arcs. When *false*, the latter signal transitions are ignored.

Previously, PrimeTime would compute a transition for the tail pin of a cell arc and propagate that transition even if this transition is not driven by the arc. This propagated transition is conservatively merged in the fanout of the arc in question, introducing additional pessimism into the timing results. The change in behavior is to eliminate this pessimism by not allowing these invalid transitions to interfere with valid propagated transitions. The latter adjustment (currently enabled by default) is disabled by this variable to assert the compatibility of timing results with releases prior to 2006.06.

To determine the current value of this variable, type **printvar timing_enable_invalid_slew_propagation_compatibility**.

SEE ALSO

printvar (2), **report_delay_calculation** (2).

timing_enable_max_capacitance_set_case_analysis

Specifies max capacitance constraint will be checked on constant pins.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable determines if max capacitance constraint is checked for constant pins. The variable can be one of two values *{true. false}* and the default is *{false}*. The value *{true}* specifies that the max capacitance will be checked for constant pins. For the current value of this variable, type **printvar timing_enable_max_capacitance_set_case_analysis**.

SEE ALSO

printvar (2), **report_constraint** -max_capacitance **(2),**

timing_enable_multiple_clocks_per_reg

Enables or disables analysis of multiple clocks that reach a single register.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable enables or disables analysis of multiple clocks that reach a register clock pin. When true (the default), all clocks reaching the register are analyzed simultaneously. When false, PrimeTime selects a random clock for analysis from among all clocks reaching a register clock pin. Do not change the value of **timing_enable_multiple_clocks_per_reg** from the default (true) unless you want this behavior.

If you set this variable to false and your design has multiple clocks per register, you should specify a clock to use with **set_data_check -clock**.

For the current value of this variable, type **printvar timing_enable_multiple_clocks_per_reg**.

SEE ALSO

create_clock (2), **create_generated_clock** (2), **set_data_check** (2), **printvar** (2).

timing_enable_preset_clear_arcs

Controls whether PrimeTime enables or disables preset and clear arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, permanently enables asynchronous preset and clear timing arcs, so that you use them to analyze timing paths. When *false* (the default), PrimeTime disables all preset and clear timing arcs.

Note that if there are any minimum pulse width checks defined on asynchronous preset and clear pins they are performed regardless of the value of this variable. Also note the the -true and the -justify options of report_timing cannot be used unless this variable is at its default value.

To determine the current value of this variable, type **printvar timing_enable_preset_clear_arcs**.

SEE ALSO

printvar (2), **report_timing** (2).

timing_enable_pulse_clock_constraints

Enables checking of pulse clock constraints.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable determines if pulse clock constraints are checked or not. The variable can be one of two values: *{true or false}*. The value *{true}* specifies that the pulse clock constraints set by set_pulse_clock_min_width, set_pulse_clock_max_width, set_pulse_clock_min_transition, and set_pulse_clock_max_transition are checked. When this variable is true the min pulse width constraints set by set_min_pulse_width command do not apply to pulse clock networks and more specifiic pulse clock constraints checked. For the current value of this variable, type **printvar timing_enable_pulse_clock_constraints**.

SEE ALSO

printvar (2), **report_constraint** -pulse_clock_min_width **(2), report_constraint** -pulse_clock_max_width **(2), report_constraint** -pulse_clock_min_transition **(2), report_constraint** -pulse_clock_max_transition **(2), set_pulse_clock_min_width, set_pulse_clock_max_width, set_pulse_clock_min_transition, set_pulse_clock_max_transition. report_pulse_clock_min_width, report_pulse_clock_max_width, report_pulse_clock_min_transition, report_pulse_clock_max_transition.**

timing_gclock_source_network_num_master_registers

The maximum number of register clock pins clocked by the master clock allowed in generated clock source latency paths.

TYPE

int

DEFAULT

1

DESCRIPTION

This variable allows the user to control the maximum number of register clock pins clocked by the master clock allowed in generated clock source latency paths. The variable does not effect the number of register traversed in a single path that do not have a clock assigned or are clocked by another generated clock that has the same primary master as the generated clock in question.

Register clock pins or transparent-D pins of registers clocked by unrelated clocks are not traversed in determining generated clock source latency paths. An unrelated clock is any clock that primary master clock differs from the generated clock who source latency paths are being computed.

To determine the current value of this variable, type **printvar timing_gclock_source_network_num_master_registers** or **echo $timing_gclock_source_network_num_master_registers**.

timing_ideal_clock_zero_default_transition

Specifies whether or not a zero transition value is assumed for sequential devices clocked by ideal clocks.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Specifies a transition value to use at clock pins of a flip-flop. If set_clock_transition command is used and the clock is ideal, the transition value will be used. This variable has no effect. If the clock transition is not set by set_clock_transition command, and the clock is ideal, then this variable will have the following effect. When *true* (the default), PrimeTime uses a zero transition value for ideal clocks. When *false*, PrimeTime uses a propagated transition value. Note that set_clock_transition will override the effect of this variable, when clock is ideal.

Note that this behavior differs from previous behavior, where PrimeTime used a propagated transition value for an ideal clock, but zero delay values at the clock pins.

To determine the current value of this variable, type **printvar timing_ideal_clock_zero_default_transition** or **echo $timing_ideal_clock_zero_default_transition**.

SEE ALSO

report_delay_calculation(2). set_clock_transition(2).

timing_include_available_borrow_in_slack

Determines whether or not PrimeTime includes available borrow time in slack.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When false (the default), the slack of a signal arriving before the latch opening edge is measured relative to the open edge and does not include available borrow time. A signal arriving during the transparent interval is considered to have a slack of zero. Violations are measured with respect to the closing latch edge.

When true, any path terminating at the data pin of a transparent latch will have positive or negative slack measured with respect to the closing transition at the latch. That is, available borrow time is considered to be a component of slack. Available borrow time is typically the duration of the active clock region minus the setup time required. A maximum time borrow set on a latch could decrease this available borrow time.

To determine the current value of this variable, type **printvar timing_include_available_borrow_in_slack** or **echo $timing_include_available_borrow_in_slack**.

SEE ALSO

printvar (2), **set_max_time_borrow** (2), **report_timing** (2).

timing_input_port_clock_shift_one_cycle

Determines whether or not paths originating at input ports are given an extra cycle to meet their timing constraints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Affects the behavior of PrimeTime when timing a path from an input port with no clocked input external delay. When *true*, paths starting at such input ports are given one extra cycle (set_multicycle_path 2) to meet timing constraints at clocked destination registers or output ports. When *false* (the default value), no extra multicycle shift is applied.

To determine the current value of this variable, type **printvar timing_input_port_clock_shift_one_cycle**.

SEE ALSO

report_timing (2).

timing_input_port_default_clock

Determines whether a default clock is assumed at input ports for which the user has not defined a clock with set_input_delay.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This Boolean variable affects the behavior of PrimeTime when the user sets an input delay without a clock on an input port. When *true* (the default value), the input delay on the port is set with respect to one imaginary clock so that the inputs are constrained. This also causes the clocks along the paths driven by these input ports to become related. Moreover, the period of this clock is equal to the base period of all these related clocks. When *false*, no such imaginary clock is assumed.

To determine the current value of this variable, type **printvar timing_input_port_default_clock**.

SEE ALSO

set_input_delay (2).

timing_keep_loop_breaking_disabled_arcs

Determines whether to keep .db inherited disabled timing arcs for static loop breaking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, enables inheriting of .db disabled timing arcs for loop breaking. When *false* (the default), do not accept .db disabled timing arcs for loop breaking.

If the .db inheritied disabled timing arcs do not break all of the loops, the default static loop breaking technique breaks the loops unless the dynamic loop breaking technique is enabled.

The .db inherited disabled timing arcs may be removed individually without affecting the other .db inheritied disabled timing arcs.

There is a difference between Design Compiler and PrimeTime where additional set_case_analysis or set_disable_timing commands do not remove .db inherited disabled timing arcs.

For this variable to take effect, you must set it before link is performed. If you set this variable after link, it has no effect.

To remove .db inherited arcs after they are accepted, they may be removed using the **remove_disable_timing** command because they are user defined.

A boolean attribute **is_db_inherited_disabled** has been added to the class timing_arcs, where true indicates an arc is a db inherited disabled arc.

To remove all .db inherited disable timing arcs for loop breaking, issue command **remove_disable_timing** [get_timing_arcs -of [get_cell *] -filter "is_db_inherited_disabled == true"]

To determine the current value of this variable, type **printvar timing_keep_loop_breaking_disabled_arcs**.

SEE ALSO

printvar (2), **report_disable_timing** (2), **remove_disable_timing** (2).

timing_non_unate_clock_compatibility

Controls whether only non-inverting clock sense is used in the non-unate case.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The backward compatible behavior for non-unate clock networks is to use only the non-inverting sense of the clock. When set to *true*, only the non-inverting sense of the clock is considered. When set to *false* (the default), both the inverting sense and non-inverting sense of the clock are analyzed simultaneously.

For the current value of this variable, type **printvar timing_non_unate_clock_compatibility**.

SEE ALSO

set_clock_sense (2).

timing_port_clock_and_data_compatibility

Disable or enable the simultaneous behavior of input port as clock and data port.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to *true*, the **set_input_delay** command has the behavior defined in PrimeTime version B-2008.12. When *false* (the default), the set_input_delay command has the behavior defined in PrimeTime version C-2009.06 and subsequent versions.

Previously, the **set_input_delay** command could also set a clock source latency at a clock port, if the ports has data sinks; the input delay could also be set only relative to clocks defined at the port. Staring in version C-2009.06, the **set_input_delay** command can be issued at a clock port, relative to clocks defined at any other port. Furthermore, the **set_input_delay** command no longer sets clock source latency.

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

```
printvar timing_port_clock_and_data_compatibility
```

SEE ALSO

printvar(2)
set_input_delay(2)
set_clock_latency(2)

timing_prelayout_scaling

Enables scaling of delay and transition times in pre-layout flow to approximate effects of mismatching driver and load signal levels.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Enables scaling of delay and transition times in pre-layout flow to approximate effects of mismatching driver and load signal levels. When this variable is set to *true* (the default value), then in pre-layout flow (without detailed parasitics) delay and transition times along net arcs are scaled to describe the same physical waveform using local trippoints and voltage level on the load cell.

No scaling is done for post-layout flow since PrimeTime measures delays on analog waveforms.

This variable is intended for obtaining backward compatibility with releases prior to 2002.09.

To determine the current value of this variable, use **printvar timing_prelayout_scaling**.

SEE ALSO

report_delay_calculation (2); **lib_thresholds_per_lib** (3).

timing_propagate_interclock_uncertainty

Enables or disables the propagation of interclock uncertainty through transparent latches in PrimeTime.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When false (the default), the interclock uncertainty is calculated for each latch-to-latch path independently, from the clock at the launch latch to the clock at the capture latch, even when latches operate in transparent mode.

When true, clock uncertainty information is propagated through each latch operating in transparent mode, as though it were a combinational element. This allows an entire sequence of latch-to-latch stages to be considered a single path for interclock uncertainty calculation, provided that time borrowing occurs at the endpoint of each intermediate stage.

Operating with this variable set to true can lead to more accurate results for designs containing transparent latches, at the cost of some CPU time and memory resources. To illustrate, consider a pipeline containing latches A, B, and C, clocked by clocks 1, 2, and 3, respectively. PrimeTime treats the paths between A and B and between B and C as distinct. In reality, however, if latch B is in transparent mode, data passes through it as though it were a combinational element. Regardless of whether interclock uncertainty has been applied between clocks 1 and 3, the default behavior is to apply the uncertainty between clocks 2 and 3 when calculating slack at latch C. It is more accurate, however, to apply the uncertainty between the clock at the path startpoint (clock 1, latch A) and the clock at the path endpoint (clock 3, latch C), if defined.

With **timing_propagate_interclock_uncertainty** set to true, the correct interclock uncertainty is applied, as though the path from latch A to latch C through the transparent latch B were a single, extended path. That is, the uncertainty is propagated through the transparent latch. To find out the startpoint of this extended path, use **report_timing -trace_latch_borrow**.

To determine the current value of this variable, type **printvar timing_propagate_interclock_uncertainty** or **echo $timing_propagate_interclock_uncertainty**.

SEE ALSO

printvar (2), **set_clock_uncertainty** (2), **report_timing** (2).

timing_propagate_through_non_latch_d_pin_arcs

Always propagate cell arcs from data pins for edge-triggered devices.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, PrimeTime always allows propagation through the cell arcs from data pins for edge-triggered devices. By default, under certain conditions PrimeTime does not allow propagation through the cell arcs from data pins of edge-triggered devices.

To determine the current value of this variable, type **printvar timing_propagate_through_non_latch_d_pin_arcs** or **echo $timing_propagate_through_non_latch_d_pin_arcs**.

Changing the value of this variable will trigger a full update_timing subsequently.

SEE ALSO

update_timing (2).

timing_propagate_through_unclocked_registers

Enables or disables propagation of the timing path through unclocked registers.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, this variable enables propagation of the timing path through unclocked registers, including edge-sensitive and level-sensitive latches. For edge-sensitive latches, the variable enables propagation through clock pin. For level-sensitive, it enables propagation through both clock pin and data pin. The default value of the variable is *false*.

By default, when some registers are not driven by a clock, the timing path breaks at the register. The path ends at the clock pin of the register, and a new path begins at the clock pin. For level-sensitive latches, the path stops at the data pin also. If a register is unclocked, so that no clock signal directly drives the register, a new path does not begin from clock pin or data pin. If the timing path is blocked, data-to-data checking cannot be performed.

Set this variable to *true* if you want to perform data-to-data checking, but your clock generation circuit contains multiple levels of registers in the fanin, leading to the related pin of a data check. A setting of *true* allows the timing path to continue through an unclocked register pin instead of being blocked.

This variable is consumed when the design is linked. Therefore, any change in value after link_design will not take effect until another re-link is invoked.

For the current value of this variable, type **printvar timing_propagate_through_unclocked_registers**.

SEE ALSO

printvar (2), **set_data_check** (2).

timing_reduce_multi_drive_net_arcs

Enables or disables the collapsing of parallel timing arcs to improve PrimeTime performance and memory utilization.

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

Designs with high-fanin, high-fanout mesh clock networks can cause significant performance degradation and explosion in memory requirements. Evidently, detailed parasitics would further exaggerate these problems. The suggested flow is to Spice the clock network and annotate clock latency and transition as ideal clock network attributes at the clock pins. To improve performance and memory requirements for clock network analysis, PrimeTime has to reduce the number of timing arcs it must consider. This is achieved by setting **timing_reduce_multi_drive_net_arcs** to *true*.

The reduction operation is performed during link; hence, the variable has to be set prior to that. Once the design is linked, modifying the value of the **timing_reduce_multi_drive_net_arcs** variable does not cause parallel timing arcs to be collapsed or restored.

Potential instances of parallel drivers are detected at design nets based on the multiplication product of the size of a net's fanin and fanout. If this product were greater than the value of the **timing_reduce_multi_drive_net_arcs_threshold** variable, the net would be considered for reduction. In order to reduce the fanin of such nets, the following criteria must be true:

1. All drivers have to be nonsequential, nonhierarchical cells.

2. All driver cells must be the same type (lib_cell).

3. All driver cells must correspondingly connect to the same nets.

For every successfully reduced net, a **PTE-046** message is issued, specifying the reduced net and the corresponding driver after the reduction. For unsuccessful attempts, a **PTE-047** message is issued to explain the reason the net drivers cannot be reduced.

Ideal clock network parameters must be set at latch clock pins in the fanout of reduced parallel buffers by using the **set_clock_transition** and **set_clock_latency** commands. An ideal clock network stipulates that the clock signal is not propagated through the parallel buffers. The cell delay accuracy of the parallel buffers or net delay accuracy of the output net of parallel buffers is not preserved. Any **report_timing** or **report_delay_calculation** commands involving these objects might show incorrect or inconsistent delays that should be overridden by ideal clock latency and transition at the register clock pins. The **check_timing** command will verify that reduced parallel buffers drive only latch clock pins with ideal clocks.

Note that the reduced cells are not physically removed from the netlist, but that no timing arcs exist to the input pins or from the output pins. Hence, flows using the **write_changes** command are not be affected. However, not having the timing arcs in and out of the collapsed cells implies that the **report_timing** command through these cells or the setting of point-to-point exceptions are completely ignored.

Flows using Standard Delay Format (SDF) annotations are accepted if ideal clock network attributes are used. SDF annotations to or from collapsed cells issue the **PTE-048** informational message noting that a particular delay annotation is ignored. Note that the remaining cell post-collapse is arbitrarily selected; and, hence, no assertion can be made as to its annotated delay. The same applies to Reduced Standard Parasitic Format (RSPF) annotations. Flows using the **write_sdf** command must account for the reduced timing arcs.

To determine the current value of the **timing_reduce_multi_drive_net_arcs** variable, type the following:

```
printvar timing_reduce_multi_drive_net_arcs
```

or

echo $timing_reduce_multi_drive_net_arcs.

SEE ALSO

check_timing (2), **printvar** (2), **report_delay_calculation** (2), **report_timing** (2), **set_annotated_transition** (2), **set_clock_latency** (2), **set_clock_transition** (2), **write_changes** (2), **write_sdf** (2), **timing_reduce_multi_drive_net_arcs_threshold** (3).

timing_reduce_multi_drive_net_arcs_threshold

Provides a threshold for the product of some net’s fanin and fanout beyond which a parallel timing arc in the net’s fanin might be reduced.

TYPE

fIintegerfP

DEFAULT

10000

DESCRIPTION

For a net, the number of timing arcs through the net is equal to the product of the net’s drivers and loads. For designs with high-fanin, high-fanout mesh clock networks, significant performance degradation and explosion in memory requirements can occur. Setting the **timing_reduce_multi_drive_net_arcs** variable improves parallel drivers reduction.

The **timing_reduce_multi_drive_net_arcs_threshold** variable provides a minimum value for the driver-load product below which the net would not be considered for reduction.

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

```
printvar timing_reduce_multi_drive_net_arcs_threshold
```

or

echo $timing_reduce_multi_drive_net_arcs_threshold.

SEE ALSO

printvar (2), **timing_reduce_multi_drive_net_arcs** (3).

timing_remove_clock_reconvergence_pessimism

Enables or disables clock reconvergence pessimism removal

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to *true*, PrimeTime removes clock reconvergence pessimism from slack calculation and minimum pulse width checks. This variable replaces the following discontinued options:

-report_clock_reconvergence_pessimism
-remove_clock_reconvergence_pessimism

of the **report_timing**, **report_constraint**, and **get_timing_paths** commands.

Clock reconvergence pessimism (CRP) is a difference in delay along the common part of the launching and capturing clock paths. The most common causes of CRP are reconvergent paths in the clock network, and different min and max delay of cells in the clock network.

CRP is independently calculated for rise and fall clock paths. You can use the variable **timing_clock_reconvergence_pessimism** to control CRP calculation with respect to transition sense. In the case of the capturing device being a level-sensitive latch two CRP values will be caculated:
• crp_open, which is the CRP corresponding to the opening edge of the latch
• crp_close, which is the CRP corresponding to the closing edge of the latch The required time at the latch will be increased by the value of crp_open and hence reduce the amount of borrowing (if any) at the latch. Meanwhile, the maximum time borrow allowed at the latch is affected by shifting the closing edge by crp_close. For more details, see the *PrimeTime User Guide: Fundamentals*.

For a more detailed description of a CRP calulation, use the **report_crpr** command.

CRP is calculated differently for minimum pulse-width checks. It is given as the minimum of (maximum rise arrival time - minimum rise arrival time) and (maximum fall arrival time - minimum fall arrival time) at the pin where the check is being made.

If the variable **si_enable analysis** is set to *true* delays in the clock network may also include delta delays resulting from crosstalk interaction. Such delays are dynamic in nature, that is, they may vary from one clock cycle to the next, causing different delay variations (either speed-up or slow-down) on the same network, but during different clock cycles.

Starting with U-2003.03 release PrimeTime only considers SI delta delays as part of the CRP calculation if the type of timing check deployed derives its data from the same clock cycle.

Similarily if dynamic annotations have been set on the design the clock delays computed using these annotations will only be used to calculate CRP if type of timing check deployed derives its data from the same clock cycle. Such dynamic annotations include, dynamic clock latency which may be specified using the **set_clock_latency** command or dynamic rail voltage which may be specified using the **set_rail_voltage** command.

In transparent-latch based designs, it is recommended that the variable **timing_early_launch_at_borrowing_latches** should be set to *false* when CRP removal (CRPR) is enabled. In this case, CRPR will apply even to paths whose startpoints are borrowing, leading to better pessimism reduction overall.

Any effective change in the value of the **timing_remove_clock_reconvergence_pessimism** variable causes full update_timing. You cannot perform one report_timing operation that considers CRP and one that does not without full update_timing in between.

For backward compatibility, the discontinued options will appear for the first few releases after they are obsoleted. However, if the design is not up to date at the time they are executed, they will only set timing_remove_clock_reconvergence_pessimism to *true*

If the design is up to date, then the command with the discontinued option fails. Since the discontinued command options only set timing_remove_clock_reconvergence_pessimism to *true*, the **-report_clock_reconvergence_pessimism** option behavior is not backward compatible. It causes slack to be removed prior to selecting the worst path. In other words, it behaves the same as the discontinued **-remove_clock_reconvergence_pessimism** option of the **report_timing**, **reyport_constraint**, and **get_timing_paths** commands. As soon as possible, update your scripts to set the **timing_remove_clock_reconvergence_pessimism** variable to *true* instead of using the discontinued options.

Limitations: CRPR does not support paths that fan out directly from clock source pins to the data pins of sequential devices. To enable support for such paths the variable **timing_crpr_remove_clock_to_data_crp** must be set to TRUE.

To turn CRP removal on:

```
   pt_shell> set timing_remove_clock_reconvergence_pessimism TRUE
TRUE
pt_shell> report_timing
```

SEE ALSO

get_timing_paths (2), **report_analysis_coverage** (2), **report_bottleneck** (2), **report_constraint** (2), **report_min_pulse_width** (2), **report_timing** (2), **set_clock_latency** (2), **set_rail_voltage** (2), **timing_crpr_threshold_ps** (3), **timing_clock_reconvergence_pessimism** (3), **timing_early_launch_at_borrowing_latches** (3), **report_crpr(2), timing_crpr_remove_clock_to_data_crp** (3), **.si_enable_analysis** (2).

timing_report_always_use_valid_start_end_points

Requires the **-from/-rise_from/-fall_from** options *from_list* objects to be valid timing startpoints and the **-to/-rise_to/-fall_to** options *to_list* objects to be valid timing endpoints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects the way the **report_timing**, **report_bottleneck** and **get_timing_path** commands interpret **-from/-rise_from/-fall_from** options *from_list* objects and **-to/-rise_to/-fall_to** options *to_list* objects.

When this variable is set to FALSE (the default), the *from_list* are interpreted as all pins found by given objects. When objects are specified with "*" or some cells name, there is a possibility that many invalid startpoints or endpoints are gotten, such as **-from [get_pins FF/*]** includes input, output, and asynchronous pins. The current behavior for PrimeTime considers these invalid startpoints or endpoints as through points and continues the path searching. Even though it is convenient to use, it is not suggested since it usually has a longer run time.

You can set this variable to TRUE to report using only valid startpoints and endpoints. It is always suggested to use input ports or register clock pins for the *from_list* objects and use output ports or register data pins for the *to_list* objects.

For the current value of this variable, type

printvar timing_report_always_use_valid_start_end_points

SEE ALSO

printvar (2)
report_timing (2)
report_bottleneck (2)
get_timing_path (2)

timing_report_maxpaths_nworst_reached

Controls maxpaths and nworst reached messages displayed during the timing report process.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Controls messages displayed during the timing report when the paths reported for a group reaches the specified max_paths and the paths reported for an endpoint reaches the specified nworst. When you set this variable to TRUE, the timing report displays messages to report whether the specified maxpaths for a group and nworst for an endpoint have been reached or not.

The messages are based on the paths before applying any filtering, such as **-slack_greater_than**. An endpoint that has exactly nworst paths is not considered as nworst reached. Therefore, you do not need to look at these endpoints for more paths. The message only displays endpoints that have more than nworst paths, so you need to increase the specified nworst to get all paths for these reported endpoints.

It is suggested that you use this variable with the **timing_report_always_use_valid_start_end_points** variable since the messages might not be accurate if invalid startpoints and endpoints are presented. You cannot use with **-true** options.

For the current value of this variable, type

printvar timing_report_maxpaths_nworst_reached

SEE ALSO

printvar (2)
report_timing (2)
get_timing_path (2)

timing_report_recalculation_status

Display progress messages during an exhaustive path-based analysis.

TYPE

string

DEFAULT

low

DESCRIPTION

This variable controls the reporting of information messages during path-based recalculation for the **report_timing** and **get_timing_paths** commands with the **-pba_mode exhaustive** option.

The number of messages varies based on the value of the variable, as follows:

- When set to *none*, PrimeTime will not show any recalculation information messages.
- When set to *low*, a message is displayed only at the end of the recalculation for each clock group.
- When set to *medium*, messages are displayed only at the beginning and end of the recalculation for each clock group.
- When set to *high*, all messages for *medium* are displayed; and, in addition, the total number of endpoints to search and the completion percentage for searching these endpoints are displayed.

Sample information messages are shown below when the variable is set to *high*.

```
Information(nworst 5, max_paths 20): recalculating group **async_default**...
Information: No paths to recalculate.
Information(nworst 5, max_paths 20): recalculating group **clock_gating_default**...
Information: No paths to recalculate.
Information(nworst 5, max_paths 20): recalculating group **default**...
Information: No paths to recalculate.
Information(nworst 5, max_paths 20): recalculating group clk...
Information: recalculated_paths 10
Information: recalculated_paths 20
Information: recalculated_paths 30
Information: finished_endpoints 3, total_endpoints 5, recalculated_paths 30
Information: finished_endpoints 5, total_endpoints 5, recalculated_paths 30
Information: Recalculated 30 paths and returned 20 paths for group 'clk'. The maxim
um number of paths recalculated at an endpoint was 10.
```

To determine the current value of this variable, type **printvar timing_report_recalculation_status**.

SEE ALSO

get_timing_paths (2),
report_timing (2).

timing_report_status_level

Controls the number of progress messages displayed during the timing report process.

TYPE

fIstringfP

DEFAULT

none

DESCRIPTION

Controls the number of progress messages displayed during the timing report process. Valid values are *none* (the default), *low*, *medium*, and *high*.

When set to *none*, no messages are displayed. When set to *low*, *medium*, or *high*, progress is reported when you use the **report_timing** and **report_constraint** commands.

The number of messages varies based on the value of the variable, as follows:

• When set to *low*, messages are displayed only at the beginning and end of the timing report.

• When set to *medium*, all messages for *low* are displayed; and, in addition, messages are displayed at the beginning of searching for each clock group.

• When set to *high*, all messages for *medium* are displayed when you use the **report_timing** command; and, in addition, the total number of endpoints to search and the completion percentage for searching these endpoints are displayed.

The report_constraint command displays status only when the **timing_report_status_level** variable is set to *high* and the **-verbose** option is used. The **report_constraint** command displays only progress status related to timing paths search.

This variable controls the display only for the timing report. Sometimes, the timing report might trigger a timing update. If you want to see the progress status for timing update, you should set the **timing_update_status_level** variable. To determine the current value of this variable, type:

printvar timing_report_status_level

or:

echo $timing_report_status_level.

SEE ALSO

printvar (2), **report_constraint** (2), **report_timing** (2); **timing_update_status_level** (3).

timing_report_unconstrained_paths

Specifies if PrimeTime searches for unconstrained paths or not when you use the **report_timing** or **get_timing_paths** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **FALSE** (the default), the **report_timing** and **get_timing_paths** commands search only for constrained paths. It runs much faster if there are lots of unconstrained paths in the design, but you are not interested in these unconstrained paths.

For backward compatiblity, you can set this variable to *TRUE*. The **report_timing** and **get_timing_paths** commands continue searching for unconstrained paths when constrained paths cannot be found. Searching unconstrained paths might have a longer run time than expected as specified by the **UITE-413** warning message.

For the current value of this variable, type

printvar timing_report_unconstrained_paths

SEE ALSO

printvar (2)
report_timing (2)
get_timing_paths (2)

timing_report_use_worst_parallel_cell_arc

Enable uniquification of paths through parallel cell arcs.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

Multiple cell arcs of the same sense can exist between the same pair of pins. In designs with large numbers of such parallel cell arcs there can often be an explosion of seemingly identical paths reported. The user can choose whether or not to report every path through parallel cell arcs.

When this variable is *false* all paths through parallel cell arcs can be reported.

When this variable is *true* only the worst path through a set of parallel cell arcs is reported. PrimeTime chooses the arc with the worst delay to determine the worst path. This variable setting has no effect in designs that have no parallel cell arcs.

SEE ALSO

get_timing_paths (2),
report_timing (2).

timing_save_pin_arrival_and_required

Specifies whether the arrival and required times of all pins are kept in memory.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, the arrival and required times of all pins of the design are kept in memory. When *false* (the default), arrival and required times are stored on an as-needed basis for the analysis the user is performing.

This variable is very similar in effect to **timing_save_pin_arrival_and_slack**, enabling the same features (particularly slack and arrival window attribute query). However, **timing_save_pin_arrival_and_required** maintains additional information is more expensive in terms of memory usage.

Users should avoid using this variable except in the specific case where the **write_sdf_constraints** forms part of the user flow, as it is the only command which requires that the additional information be stored at all pins. If the **write_sdf_constraints** command is used while this variable is *false*, it will be set to *true* automatically and an informational message issued.

To determine the current value of this variable, type **printvar timing_save_pin_arrival_and_required**.

SEE ALSO

printvar (2), **timing_save_pin_arrival_and_slack** (3).

timing_save_pin_arrival_and_slack

Specifies whether the slacks of all pins are kept in memory.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When *true*, slacks of all pins of the design are kept in memory. When *false* (the default), slacks are preserved only for endpoints of the design.

To query slack attributes or arrival window attributes on pins that are not endpoints of the design, set this variable to *true*.

To determine the current value of this variable, type **printvar timing_save_pin_arrival_and_slack**.

SEE ALSO

printvar (2), **report_timing** (2).

timing_si_exclude_delta_slew_for_transition_constraint

Specifies delta slew to be excluded from maximum and minimum transition constraint checks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable determines if delta slew is to be excluded from maximum and minimum transition constraint checks. The variable can be one of two values: *{true or false}* (default). The value *{true}* specifies that the delta slew is excluded for maximum and minimum transition constraint checks. For the current value of this variable, type **printvar timing_si_exclude_delta_slew_for_transition_constraint**.

SEE ALSO

printvar (2), **report_constraint** -max_transition **(2), report_constraint** -min_transition **(2).**

timing_port_clock_and_data_compatibility

Disable or enable the simultaneours behavior of input port as clock and data port,

TYPE

Boolean

DEFAULT

false

DESCRIPTION

PrimeTime allows an input port to behave simultaneously as a clock and data port. You can use the timing_simultaneous_clock_data_port_compatibility variable to enable or disable the simultaneous behavior of the input port as a clock and data port. When this variable is false, the default, simultaneous behavior is enabled and you can use the set_input_delay command to define the timing requirements for input ports relative to a clock. In this situation, the following applies:

> If you specify the set_input_delay command relative to a clock defined at the same port and the port has data sinks, the command is ignored and an error message is issued. There is only one signal coming to port, and it cannot be at the same time data relative to a clock and the clock signal itself.
> If you specify the set_input_delay command relative to a clock defined at a different port and the port has data sinks, the input delay is set and controls data edges launched from the port relative to the clock.
> Regardless of the location of the data port, if the clock port does not fanout to data sinks, the input delay on the clock port is ignored and you receive an error message.
> When you set the timing_simultaneous_clock_data_port_compatibility variable to true, the simultaneous behavior is disabled and the set_input_delay command defines the arrival time relative to a clock. In this situation, when an input port has a clock defined on it, PrimeTime considers the port exclusively as a clock port and imposes restriction on the data edges that are launched. PrimeTime also prevents setting input delays relative to another clock.
> To determine the current value of this variable, type **printvar timing_port_clock_and_data_compatibility**.

SEE ALSO

printvar (2), **set_input_delay** (2). **set_clock_latency** (2).

timing_slew_propagation_mode

Specifies how slew is to be propagated through the circuit.

TYPE

String

DEFAULT

worst_slew

DESCRIPTION

This variable specifies how slew is to be propagated through the circuit. Allowed values are *worst_slew* (the default), which causes PrimeTime to propagate the worst slew selected from the inputs; and *worst_arrival*, which causes PrimeTime to select the slew of the input with the worst arrival time, from multiple inputs propagated from the same clock domain. The propagated slew affects the delay of a cell arc or net.

In *worst_slew* mode, at a pin where multiple timing arcs meet (or merge), PrimeTime computes the slew per driving arc at the pin, then selects the worst slew value at the pin to propagate along. Note that the slew selected might not be from the input that contributes to the worst path, so the calculated delay from the merge pin could be pessimistic.

In *worst_arrival* mode, PrimeTime selects and propagates the slew of the input with the worst arrival time, from multiple inputs propagated from the same clock domain; PrimeTime selects and propagates different slews from each clock domain.

Minimum slew propagation is similar to maximum slew propagation; that is, minimum slew is selected based on the input with the best delay at the merge point. Maximum slews (rise and fall) are used to calculate maximum arc delays, while minimum slews (rise and fall) are used to calculate minimum arc delays. This results in more accurate path delay propagation. Because more than one slew can be propagated at a pin, the delay of an arc is not fixed; it depends on the input slew being propagated. Also, because both max and min slews are propagated, an arc has both min and max delays. This is equivalent to analyzing the design in the on-chip variation mode. PrimeTime issues an informational message and echos the exact **set_operating_condition** command used. If the design is set in single operating conditions mode while the variable is set to worst_arrival, PrimeTime automatically sets the operating conditions of the design in on-chip variations mode with both the best-case and the worst-case operating condtion being set to the user-specified single-operating condition mode.

This variable affects several other commands in PrimeTime and has inherent limitations. One of the limitations is that writing SDF for a design in worst_arrival mode is not accurate. In this mode, PrimeTime has multiple delays per arc, which cannot be represented in SDF. PrimeTime writes out the most pessimistic delay value available for any arc. Also, the report_delay_calculation command should be used only with -from_rise_trans and -from_fall_trans switches set so as to show the delay calculation for those slews that are specified for the source pin.

For the current value of this variable, type **printvar timing_slew_propagation_mode**.

SEE ALSO

printvar (2), **set_operating_condition** (2),
timing_slew_propagation_worst_arrival_fanout_sensitivity (3).

timing_slew_threshold_scaling_for_max_transition_compatibility

Specifies the compatibility mode for timing slew threshold scaling.

TYPE

fIBooleanfP

DEFAULT

false

DESCRIPTION

The scaling of transition time for slew threshold is on by default in Z-2006.12. The user needs to set this variable to true to revert to the behavior prior to Z-2006.12.

SEE ALSO

set_max_transition (2).

timing_update_default_mode

Determines whether **update_timing** performs incremental or complete timing analysis.

TYPE

string

DEFAULT

incremental

DESCRIPTION

This variable determines whether the **update_timing** command performs incremental or complete timing analysis. If the value of this variable is set to *incremental* (the default), the **update_timing** command performs incremental timing analysis, which uses an efficient algorithm that requires minimal computation effort and updates existing timing analysis information only where needed. If **timing_update_default_mode** is set to *full*, the default behavior of **update_timing** is the same as **update_timing -full**, in which the **update_timing** command performs complete timing analysis from the beginning.

For the current value of this variable, type **printvar timing_update_default_mode**.

SEE ALSO

printvar (2), **update_timing** (2).

timing_update_effort

Controls the computational effort (in CPU time) and memory usage for the fast timing update algorithm in PrimeTime.

TYPE

string

DEFAULT

medium

GROUP

timing_variables

DESCRIPTION

Controls the computational effort (in CPU time) and memory usage for the fast timing update algorithm in PrimeTime. Allowed values are as follows:

* *low*: The computational effort is low (that is, **report_timing** is fast), but the memory usage is not bounded and can increase significantly if the number of changes is very large.

* *medium* (the default): The computational effort is low (that is,**report_timing** is fast), but the memory usage is bounded by 10% over the memory usage for initial timing. If this bound is not sufficient to accommodate all of your changes, PrimeTime issues an informational message and automatically switches to a less efficient algorithm that is more conservative in the memory usage. In this case, you might need to change the value to *low*. However, even with this small 10% bound, PrimeTime can accommodate a relatively large number of changes. Therefore, it is unlikely that you will need to change this default value.

* *high*: The computational effort is high (that is report_timing becomes slow) but there is no increase in the memory used over that for the initial timing of the design.

When a design is retimed after a change, the algorithm reuses a portion of the computation done for the initial timing. For example, if a design was loaded and timed using the commands **update_timing** or **report_timing**, and the capacitance on a port was changed using the command **set_capacitance**, the effort spent in the execution of a subsequently issued **report_timing** command is smaller than that for the first issued **report_timing** or **update_timing** commands.

For the current value of this variable, type **printvar timing_update_effort**.

SEE ALSO

printvar (2), **report_timing** (2), **update_timing** (2).

timing_update_status_level

Controls the number of progress messages displayed during the timing update process.

TYPE

fIstringfP

DEFAULT

none

GROUP

timing_variables

DESCRIPTION

Controls the number of progress messages displayed during the timing update process. Allowed values are *none* (the default), *low*, *medium*, or *high*.

When set to *none*, no messages are displayed. When set to *low*, *medium*, or *high*, the progress of the timing update is reported for an explicit update (using the **update_timing** command) or for an implicit update invoked by another command (for example, **report_timing**) that forces a timing update. The number of messages varies based on the value of the variable, as follows:

When set to *low*, messages are displayed only at the beginning and the end of the update.

When set to *medium*, all messages for *low* are displayed, and in addition, messages are displayed at the beginning and at the end of the additional intermediate timing update steps constant propagation, delay calculation, and slack computation.

When set to *high*, all messages for *medium* are displayed; in addition, messages for the delay calculation step show the completion percentage for large designs, and messages for the slack computation step show the groups for which the computation is made.

To determine the current value of this variable, type **printvar timing_udpate_status_level** or **echo $timing_update_status_level**.

SEE ALSO

printvar (2), **report_timing** (2), **update_timing** (2).

timing_use_zero_slew_for_annotated_arcs

Allows disabling of the slew calculation to enhance performance in a pure SDF flow.

TYPE

list

DEFAULT

auto

DESCRIPTION

This variable allows the user to sacrifice slew calculation for performance in an SDF flow. The valid variable values are *always*, *auto* or *never*.

When set to *always*, a zero value is used for transition time on the load pins of fully delay annotated arcs. Fully annotated arcs are those with values for both rise and fall either read from an SDF file, or set with the command **set_annotated_delay**.

Should there exist blocks of arcs that are not annotated, delay is estimated using the best available slew at the inputs. This functionality requires **timing_slew_propagation_mode** to be set to *worst_slew*, as other modes of slew propagation are not supported within these blocks.

The default value is *auto*, which allows the automatic switching to the SDF flow if more than 95 percent of delay arcs on a design have annotated values.

To avoid usage of the SDF flow, set **timing_use_zero_slew_for_annotated_arcs** to *never*.

For the current value of this variable, type **printvar timing_use_zero_slew_for_annotated_arcs**.

SEE ALSO

printvar (2), **read_sdf** (2), **timing_slew_propagation_mode** (3)

true_delay_prove_false_backtrack_limit

Specifies the number of backtracks to be used by **report_timing -true** in searching for false paths.

TYPE

int

DEFAULT

1000

GROUP

timing_variables

DESCRIPTION

An integer variable that specifies the number of backtracks to be used by **report_timing -true** in searching for false paths; **-1** specifies unlimited backtracking. The default is 1000. This is one of a pair of variables; the other is **true_delay_prove_true_backtrack_limit**.

In almost all cases, PrimeTime finds the longest true path fairly quickly if a low backtrack limit is used. However, there are some designs that require a high backtrack limit to find the real longest path, but these designs are rare. The default backtrack limit of 1000 provides a good compromise between runtime and accuracy.

When it reaches the backtrack limit specified by **true_delay_prove_true_backtrack_limit**, the algorithm stops searching for true paths and saves the longest true path found. This path is proven true, but because the search reached the backtrack limit there can be longer true paths that were not found. Therefore,the proven true path provides a lower bound on the delay of the design. The algorithm then goes into a second phase in which it tries to prove paths false, starting with the structurally longest path and working downward. This second phase provides a tighter upper bound on the true delay of the design.

The prove false phase can also reach the backtrack limit specified by **true_delay_prove_false_backtrack_limit**. When both phases are complete, the true delay is between the lower (proved-true) value and the upper (proved-false) value.

For the current value of this variable, type **printvar true_delay_prove_false_backtrack_limit**.

SEE ALSO

printvar (2), **report_timing** (2), **true_delay_prove_true_backtrack_limit** (3).

true_delay_prove_true_backtrack_limit

Specifies the number of backtracks to be used by **report_timing -true** in searching for true paths.

TYPE

int

DEFAULT

1000

GROUP

timing_variables

DESCRIPTION

An integer variable that specifies the number of backtracks to be used by **report_timing -true** in searching for true paths; **-1** specifies unlimited backtracking. The default is 1000. This is one of a pair of variables; the other is **true_delay_prove_false_backtrack_limit**.

In almost all cases, PrimeTime can find the longest true path fairly quickly if a low backtracking limit is used. However, there are some designs that require a high backtrack limit to find the real longest path, but these designs are rare. The default backtrack limit of 1000 provides a good compromise between runtime and accuracy.

When it reaches the backtrack limit specified by **true_delay_prove_true_backtrack_limit**, the algorithm stops searching for true paths and saves the longest true path found. This path is proven true, but since the search reached the backtrack limit there can be longer true paths that were not found. Therefore, the proven true path provides a lower bound on the delay of the design. The algorithm then goes into a second phase in which it tries to prove paths false, starting with the structurally longest path and working downward. This second phase provides a tighter upper bound on the true delay of the design.

The prove false phase can also reach the backtrack limit specified by **true_delay_prove_false_backtrack_limit**. When both phases are complete, the true delay is between the lower (proved-true) value and the upper (proved-false) value.

For the current value of this variable, type **printvar true_delay_prove_true_backtrack_limit**.

SEE ALSO

printvar (2), **report_timing** (2), **true_delay_prove_false_backtrack_limit** (3).

variation_analysis_mode

Specifies the scope of variations analysis performed during a subsequent timing update.

TYPE

String

DEFAULT

default

DESCRIPTION

The setting of this variable encodes the scope of the subsequent graph-based static timing analysis performed during update_timing, provided the variation_enable_analysis is set to TRUE. If the latter variable were FALSE, this variable has no effect.
The valid choices of effort level are default, detailed_clock_timing, and path_based_only.
To determine the current value of this variable, type **printvar variation_analysis_mode**.

SEE ALSO

update_timing (2), **variation_enable_analysis** (3).

variation_derived_scalar_attribute_mode

Enables get_attribute command to return statistical timing attributes of timing_path and timing_point collections in VASTA.

TYPE

String

DEFAULT

quantile

DESCRIPTION

This variable is used to enable or disable statistical timing attributes as the return values of get_attribute command for timing_path and timing_point collections. If it is set to mean or quantile, the arrival, slack, required, or transition values returned from command get_attribute will be replaced with the mean or quantile values of corresponding variation_arrival, variation_slack, variation_required, or variation_transition attributes. The default value is quantile in variation-aware analysis. Setting this variable to none will disable statistical attributes, and get_attribute will return the corner values. If variation-aware analysis is turned off, get_attribute will always return the corner values regardless of the value of this variable.
Commands sort_collection and filter_collection will be affected if this variable is set to mean or quantile while related timing attributes are used in the sorting or filtering schemes. This feature allows to create custom timing_path collections that are sorted/filtered by statistical timing attributes.
The default quantile value can be defined by command set_variation_analysis_mode. To determine the current value of this variable, type **printvar variation_derived_scalar_attribute_mode** or **echo $variation_derived_scalar_attribute_mode**.

SH EXAMPLES

The following example replaces the returned arrival value of get_attribute from corner value to mean of variation_arrival.

```
pt_shell> set variation_derived_scalar_attribute_mode none
none
pt_shell> get_attribute $path arrival
0.25
pt_shell> set variation_derived_scalar_attribute_mode mean
mean
pt_shell> get_attribute $path arrival
0.22
```

This example creates a new timing_path collection by sorting an existing collection by statistical quantile of variation_slack.

```
pt_shell> set path [get_timing_path -nworst 10]
_sel22
pt_shell> set variation_derived_scalar_attribute_mode quantile
quantile
pt_shell> set stat_path [sort_collection $path "arrival"]
_sel23
```

SEE ALSO

variation_enable_analysis (3), **set_variation_analysis_mode** (2), **get_attribute** (2), **sort_collection** (2), **filter_collection** (2).

variation_enable_analysis

Enables statistical variation analysis for PrimeTime.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable is used to enable or disable statistical variation analysis. Set this variable to true to enable statistical variation, false to disable it. When set to true PrimeTime attempts to get the appropirate license. If it gets the license then the variable is set to true. If it cannot get the license then the variable is left at false.
To determine the current value of this variable, type **printvar variation_enable_analysis** or **echo $variation_enable_analysis**.

SEE ALSO

set_variation_library (2), **create_distribution** (2), **set_variation** (2), **variation_analysis_mode** (3).

variation_pba_use_worst_parasitics

Enables use of worst parasitic corner for statistical path based analysis (PBA).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable is used to enable or disable use of eorst parasitic corner per path for statistical path based analysis. Set this variable to true to enable use of worst parasitics, false to disable it. When set to true, PrimeTime-VX attempts to calculate worst parasitic corner (worst slack corner) every-time a PBA command is issued. The parasitic variations are treated as constants at the computed corner values and statistical analysis with device variations is carried out. Worst parasitic corner is computed only if the design has variation aware parasitics read using **read_parasitics -keep_variations** and the auto-correlation for the parasitic variation parameters is 1. Cross-correlation values of only +1/-1 between parasitic variation parameters are honored to determine the worst corner. All other cross-correlations values are ignored (considered to be 0) for determining the worst corner.
To determine the current value of this variable, type **printvar variation_pba_use_worst_parasitics** or **echo $variation_pba_use_worst_parasitics**.

SEE ALSO

read_parasitics(2), set_variation_correlation(2).

variation_report_timing_increment_format

Controls the display of report_timing increments for variation-aware timing paths.

TYPE

String

DEFAULT

effective_delay

DESCRIPTION

This variable affects the display of paths which have been recalculated within the context of a variation-aware timing analysis. The transition times, delays and arrival times associated with these paths are statistical in nature. This variable lets the user configure the display of the delays appearing in the increment column (labelled Incr).
Allowed values are *effective_delay* (the default) and *delay_variation*. When set to *effective_delay*, each increment displayed equals the scalar difference between the arrival values appearing in the Path column. When set to *delay_variation*, the quantile value (or mean value) of the statistical increment is displayed instead, according to the **variation_derived_scalar_attribute_mode** variable.
The arrival times and transitions are also displayed using scalar representations of their underlying distributions. These too obey the **variation_derived_scalar_attribute_mode** variable.
To determine the current value of this variable, type **printvar fBvariation_report_timing_increment_format**.

SEE ALSO

variation_enable_analysis (3), **variation_analysis_mode** (3), **variation_derived_scalar_attribute_mode** (3), **report_timing** (2).

write_script_include_library_constraints

This variable is used to control whether constraints set on library objects are written to script output by **write_script** and **write_sdc**.

TYPE

fIBooleanfP

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This variable is used to control whether constraints set on library objects are written to script output by **write_script** and **write_sdc**. When *true* (the default), these constraints are written. The only constraints which are written are those attached to objects in libraries which are in use by the current design. Currently, only constraints created with **set_disable_timing** are written.
To determine the current value of this variable, type **printvar write_script_include_library_constraints** or **echo $write_script_include_library_constraints**.

SEE ALSO

printvar (2), **set_disable_timing** (2), **write_script** (2), **write_sdc** (2).

write_script_output_lumped_net_annotation

Determines whether or not the **write_script** command outputs lumped network annotations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When true, the **write_script** command outputs lumped network annotations. When false (the default), **write_script** does not output lumped network annotations, because they are not valid equivalents of detailed network annotations made by the **read_parasitics** command. The lumped network annotations are total capacitance (set by **set_load**) and total resistance (set by **set_resistance**). Set this variable to true if you need lumped annotations to be included in the **write_script** output.

If you want intentional **set_resistance**, **set_load**, and **set_capacitance** annotations, it is preferable to include them in a separate Tcl script rather than in a Synopsys Design Constraints (SDC) file.

To determine the current value of this variable, type **printvar write_script_output_lumped_net_annotation**.

SEE ALSO

read_parasitics (2), **set_load** (2), **set_resistance** (2), **write_script** (2).